PANEL '81 EXPODATA

Julio 12

VIII CONFERENCIA LATINOAMERICANA DEL CLEI

Centro Latinoamericano de Estudios en **Informática**

12° Jornadas Argentinas de **Informática e Investigación Operativa**

BUENOS AIRES MARZO, 1981

Decreto de Interés Nacional Nro. 2746

Decreto de Interés Municipal Nro. 7316

PATROCINADO POR:

IBI, Oficina Intergubernamental para la Informática

CON LOS AUSPICIOS DE:

Secretaría de Estado de Ciencia y Tecnología, SECYT

Subsecretaría de Informática, Secretaría de Planeamiento, Presidencia de la Nación

Subsecretaría de la Función Pública, Presidencia de la Nación

Municipalidad de la Ciudad de Buenos Aires

COMITE DE HONOR

BRIGADIER (R) D. OSVALDO ANDRES CACCIATORE
Intendente de la Municipalidad de la Ciudad de Buenos Aires

DR. FERMIN GARCIA MARCOS
Secretario de Estado de Ciencia y Tecnología

BRIGADIER D. JOSE MIRET
Secretario de Planeamiento de la Presidencia de la Nación

DR. FERMIN BERNASCONI
Director General del IBI

COMODORO (R) D. OSCAR G. VELEZ
Subsecretario de Informática

CAPITAN DE NAVIO (R) I.M. MARIO A. VIDIELLA
Subsecretario de la Función Pública

AUTORIDADES DEL PANEL

Presidente:

GUSTAVO A. POLLITZER

Tesorero:

ISAAC BLEGER

COMITE DE PROGRAMA

FABIO ANTONIOLI P. (Venezuela)
JOSE LUIS BENZA (Paraguay)
TEREZINHA CHAVEZ (Brasil)
ANTONIO DONADIO MEDAGLIA (México)
LEONARDO ELIZALDE CALLE (Ecuador)
RAFAEL O. FONTAO (Argentina)
LUIS MEDINA (Bolivia)
HECTOR PEREZ L. (Chile)
FELIX PIMENTEL (Uruguay)
JOSE PORTILLO CAMPBELL (Perú)
ARNOLD SCHIEMANN (Colombia)

COMITE ACADEMICO

Presidentes:

HORACIO BOSCH
ISIDORO MARIN

Coordinador:

CARLOS TOMASSINO

Coordinadores de Sesión:

ROBERTO ANTELO
LEOPOLDO CARRANZA
CLAUDIO DI VEROLI
CHARLES FRANCOIS
EDUARDO JORDAN
EDUARDO LOSOVIZ
GUILLERMO MARSHALL
RUBEN MICHELSON
RODOLFO J. NAVEIRO
ANTONIO A. QUIJANO
HORACIO C. REGGINI
ROBERTO SCHTEINGART
GUIDO VASALLO

COMITE EJECUTIVO

Presidente:

ARAEL NAGGI BROWN

JORGE BRUNIARD
JORGE CASSINO
IGNACIO HERNANDEZ
EDUARDO LOSOVIZ
COSME NARANJO
ESPEDITO PASSARELLO
ARTURO PEREZ ARRIBILLAGA
DARIO PICCIRILLI
RAUL O. RISOLI

AUTORIDADES DEL CLEI

Presidente:

VICTOR YOCKTENG (Perú)

Consejo Directivo:

JUAN CARLOS ANSELMI (Uruguay)
JULIAN ARAOZ (Venezuela)
RAFAEL FONTAO (Argentina)
FABIAN GAMBOA (Ecuador)
AMILCAR MORALES (Chile)
CARLOS J. LUCENA (Brasil)
CORRADO VALLET (Bolivia)

Secretario Ejecutivo:

ALDO MIGLIARO (Chile)

AUTORIDADES DE SADIO

Presidente:

HECTOR MONTEVERDE

Vicepresidente:

RODOLFO J. NAVEIRO

Secretario:

VALERIO J. YACUBSOHN

Tesorero:

ERNESTO UGARTE REY

Vocales:

JUAN C. ANGIO
LEOPOLDO CARRANZA
OSVALDO GOSMAN
EDUARDO LOSOVIZ

Vocales Suplentes:

GRACIELA CHORNY
ANGEL A. OLMOS

Consejo Asesor:

JORGE BASSO DASTUGUE
FERMIN BERNASCONI
ISIDORO MARIN
OSVALDO MOLINA
HUGO MORUZZI
ENRIQUE LECHNER

AGRADECEMOS EL APOYO DE:

- Coronel (R) JULIO CESAR ABRAMOFF
  Organización y Sistemas de Entel

- Ing. JORGE LUIS FERRANTE
  Secretaría de Ciencia y Tecnología

- Dr. FEDERICO FRISCHKNECHT

- Dr. RICHARD HERBERT
  Agregado Científico Consulado Británico

- Ing. JOSE LUIS MENDIBURU
  Centro Unico de Procesamiento Electrónico de Datos

- Sr. SERGIO ORCE
  IBI - Oficina Intergubernamental para la Informática

- Sr. SEBASTIAN PICCONE
  Banco Nacional de Desarrollo

- Dr. FERNANDO PIERA
  IBI - Oficina Intergubernamental para la Informática

- Sr. PHILIP W. PILLSBURY
  Embajada de los Estados Unidos de América

- Dr. TITO SUTER
  Comisión Nacional de Energía Atómica

- Ing. JULIO CESAR YOUNG OLIVER
  Instituto Nacional de Tecnología Industrial

COLABORACIONES:

- Banco de la Nación Argentina
- Ciccone Hnos. y Lima
- Sistronic
- Comisión Nacional de Energía Atómica
- Consejo de Investigaciones de Noruega
- Dirección General Registro Automático de Datos (DIGRAD)
- Editorial El Ateneo
- ENCOTEL
- ENTEL
- Lotería de Beneficiencia Nacional y Casinos
- Municipalidad de la Ciudad de Buenos Aires
- Secretaría de Planeamiento Presidencia de la Nación
- Subsecretaría de Informática
- Subsecretaría de la Función Pública
- 3 M Argentina
- Universidad de Belgrano, Facultad de Tecnología

EXPODATA

- AUTOM S.R.L.
- COASIN COMPUTACION
- COMDATA S.A.
- COMPUCORP S.A.
- HEWLETT PACKARD ARGENTINA S.A.
- IBM ARGENTINA S.A.
- NOVADATA
- OLIVETTI ARGENTINA S.A.
- S.A.C.O.M.A
  CENTRO DE COMPUTACION DE DATOS
- SELENIA S.p.A.
- TECNOBETON S.A.
- THINKERCORP S.R.L.
- UNIVAC DIVISION DE SPERRY RAND ARGENTINA S.A.

PARTICIPACION INSTITUCIONAL

Patrocinante Extraordinario

IBM ARGENTINA S.A.

Patrocinantes

BANCO NACIONAL DE DESARROLLO

CENTRO UNICO DE PROSESAMIENTO ELECTRONICO DE DATOS

DIRECCION GENERAL IMPOSITIVA

INSTITUTO NACIONAL DE TECNOLOGIA INDUSTRIAL

PROCEDA S.A.

UNIVERSIDAD ARGENTINA JOHN F. KENNEDY

Participantes

COMISION NACIONAL DE ENERGIA ATOMICA

DINERS CLUB ARGENTINA SAC y DET

EMPRESA NACIONAL DE TELECOMUNICACIONES

ESCUELA SUPERIOR TECNICA DEL EJERCITO

FAFIDE SACIF

NOVADATA - GASES INDUSTRIALES SATIC

PROMEX S.A.

SHELL CAPSA

UNIVERSIDAD DE BUENOS AIRES

Adherentes

ASTILLEROS Y FABRICAS NAVALES DEL ESTADO

ALPARGATAS SAIC

ASTARSA

BANCO CENTRAL DE LA REPUBLICA ARGENTINA

BANCO COMERCIAL DEL NORTE

BANCO FRANCES DEL RIO DE LA PLATA S.A.

BANCO DE GALICIA Y BUENOS AIRES

BANCO POPULAR ARGENTINO

BULL ARGENTINA S.A.C.I.

CARGILL S.A.C.I.

CASPI

COMPAÑIA FINANCIERA RIOMAR S.A.

DETOITE PLENDER HASKINS & SELLS

ESSEX S.A.I.C.

FACULTAD DE MATEMATICA APLICADA UCALP

LOMA NEGRA C.I.A.S.A.

NOBLEZA PICARDO S.A I.C. y F.

OFICINA DE AUDITORIA F.W. SIBILLE

YACIMIENTOS PETROLIFEROS FISCALES

PARTICIPACION INDIVIDUAL

Adherente Individual

EDUARDO CASTIÑEIRA

LILIA ELBA CHATRUC

ARMANDO EDUARDO DE GIUSTI

FRANCISCO JAVIER DIAZ

DANIEL HERMINIO FORTUNATI

LUIS RICARDO LOPEZ MATEO

ALFREDO RAUL MUÑIZ MORENO

AMALIA LIDIA PALACIOS

ALFREDO DANIEL POLINI

MARIO RAPIZARDI

GUSTAVO HECTOR ROSSI

PATRICIA LIA ZABALZA

PRESENTACION

La reunión conjunta PANEL 81/12 JAIIO incluye en su programación el desarrollo de Sesiones de exposición y discusión de trabajos, Conferencias a cargo de especialistas invitados, Mesas Redondas de intercambio y discusión de temas específicos y Seminarios de actualización.

En los Anales que aquí presentamos se incluye en sus dos tomos el material de consulta para la primera de dichas actividades, esto es, los textos de los trabajos, agregados en capítulos temáticos correspondientes a las sesiones en que van a ser tratados.

Es nuestra intención, y queda establecido el compromiso, editar próximamente un tercer tomo de los Anales, al que se incorpore la información escrita resultante de las restantes actividades a llevarse a cabo en el Congreso: Conferencias, Mesas Redondas y Seminarios. Consideramos que tal información será de especial interés, particularmente para aquellas personas que no hayan intervenido personalmente en la reunión.

Nos hemos esforzado para tratar que los trabajos aquí incluidos aparezcan libres de errores; no obstante, pedimos a sus autores que nos informen sobre eventuales deficiencias de importancia, para así resolverlas en una fe de erratas a aparecer en el tercer tomo previsto.

Por último y más importante, expresamos nuestro reconocimiento a todas aquellas personas y entidades que hicieron posible la edición de estos Anales; gracias.

JUAN IGNACIO HERNANDEZ
Editor Anales Panel 81/12 Jaiio

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

CAPITULO A

CIENCIA DE COMPUTACION

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

ALGORITHMIC INFORMATION THEORY

TEORIA DE LA INFORMACION ALGORITMICA

Gregory J. Chaitin
IBM Thomas J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York 10598, U.S.A.

Abstrac: We give a brief introduction to algorithmic information theory, a new field which combines ideas from information theory and from the theory of algorithms. We also briefly discuss the new light thrown by algorithmic information theory on foundational issues in the theories of probability and mathematical logic.

Resumen: Damos aquí una breve introducción a la teoría de la información algorítmica, un nuevo campo de estudios que combina ideas de la teoría de la información y de la teoría de los algorítmos. También nos referimos brevemente a la manera en que la teoría de la información algorítmica aclara ciertos problemas de base de las teorías de la probabilidad y de la lógica matemática.

The Shannon entropy concept of classical information theory [1] is an ensemble notion; it is a measure of the degree of ignorance concerning which possibility holds in an ensemble with a given a priori probability distribution:

$$H(p_1,\ldots,p_n) \equiv -\sum_{k=1}^{n} p_k \log_2 p_k.$$

In algorithmic information theory the primary concept is that of the *information content* of an individual object, which is a measure of how difficult it is to specify or describe how to construct or calculate that object. This notion is also known as *information-theoretic complexity*. For introductory expositions, see [2-4]. For the necessary background on computability theory and mathematical logic, see [5-7]. For a more technical survey of algorithmic information theory and a more complete bibliography, see [8]. See also [9].

The original formulation of the concept of algorithmic information is independently due to R. J. Solomonoff [10], A. N. Kolmogorov [11], and G. J. Chaitin [12]. The information content $I(x)$ of a binary string x is defined to be the size in bits (binary digits) of the smallest program for a canonical universal computer U to calculate x. (That the computer U is universal means that for any other computer M there is a prefix μ such that the program μp makes U do exactly the same computation that the program p makes M do.) The *joint information* $I(x, y)$ of two strings is defined to be the size of the smallest program that makes U calculate both of them. And the *conditional* or *relative information* $I(x \mid y)$ of x given y is defined to be the size of the smallest program for U to calculate x from y. The choice of the standard computer U introduces at most an $O(1)$ uncertainty in the numerical value of these concepts. ($O(f)$ is read "order of f" and denotes a function whose absolute value is bounded by a constant times f.)

With the original formulation of these definitions, for most x one has

$$I(x) = |x| + O(1) \tag{1}$$

(here $|x|$ denotes the length or size of the string x, in bits), but unfortunately

$$I(x, y) \leq I(x) + I(y) + O(1) \tag{2}$$

only holds if one replaces the $O(1)$ error estimate by $O(\log I(x)I(y))$.

Chaitin [13] and L. A. Levin [14] independently discovered how to reformulate these definitions so that the subadditivity property (2) holds. The change is to require that the set of meaningful computer programs be an instantaneous code, that is, that no program be a prefix of another. With this modification, (2) now holds, but instead of (1) most x satisfy

$$\begin{aligned} I(x) &= |x| + I(|x|) + O(1) \\ &= |x| + O(\log |x|). \end{aligned}$$

Moreover, in this theory the decomposition of the joint information of two objects into the sum of the information content of the first object added to the relative information of the second one given the first, has a different form than in classical information theory. In fact, instead of

$$I(x, y) = I(x) + I(y|x) + O(1), \tag{3}$$

one has

$$I(x, y) = I(x) + I(y|x, I(x)) + O(1). \tag{4}$$

That (3) is false follows from the fact that $I(x, I(x)) = I(x) + O(1)$ and $I(I(x)|x)$ is unbounded. This was noted by Chaitin [13] and studied more precisely by R. M. Solovay [13, p. 339] and P. Gač [15].

Two other concepts of algorithmic information theory are *mutual* or *common information* and *algorithmic independence*. Their importance has been emphasized by T. L. Fine [9, p. 141]. The mutual information content of two strings is defined as follows:

$$I(x : y) \equiv I(x) + I(y) - I(x, y).$$

In other words, the mutual information of two strings is the extent to which it is more economical to calculate them together than to calculate them separately. And x and y are said to be algorithmically independent if their mutual information $I(x : y)$ is essentially zero, that is, if $I(x, y)$ is approximately equal to $I(x) + I(y)$. Mutual information is symmetrical, i.e., $I(x : y) = I(y : x) + O(1)$. More important, from the decomposition (4) one obtains the following two alternative expressions for mutual information:

$$\begin{aligned} I(x : y) &= I(x) - I(x|y, I(y)) + O(1) \\ &= I(y) - I(y|x, I(x)) + O(1). \end{aligned}$$

Thus this notion of mutual information, although it applies to individual objects rather than to ensembles, nevertheless shares many of the formal properties of the classical version of this concept.

Up to this time there have been two principal applications of algorithmic information theory: (a) to provide a new conceptual foundation for probability theory and statistics by making it possible to rigorously define the notion of a *random sequence,* and (b) to provide an information-theoretic approach to metamathematics and the limitative theorems of mathematical logic. A possible application to theoretical mathematical biology is also mentioned below.

A random or patternless binary sequence x_n of length n may be defined to be one of maximal or near maximal complexity, that is, one whose complexity $I(x_n)$ is not much less

than n. Similarly, an infinite binary sequence x may be defined to be random if its initial segments x_n are all random finite binary sequences. More precisely, x is random if and only if

$$\exists c \forall n [I(x_n) > n - c]. \tag{5}$$

In other words, the infinite sequence x is random if and only if there exists a c such that for all positive integers n, the algorithmic information content of the string consisting of the first n bits of the sequence x, is bounded from below by $n - c$. Similarly, a *random real number* may be defined to be one having the property that the base-two expansion of its fractional part is a random infinite binary sequence.

These definitions are intended to capture the intuitive notion of a lawless, chaotic, unstructured sequence. Sequences certified as random in this sense would be ideal for use in Monte Carlo calculations [16], and they would also be ideal as one-time pads for Vernam ciphers or as encription keys [17]. Unfortunately, as we shall see below, it is a variant of Gödel's famous incompleteness theorem that such certification is impossible. It is a corollary that no pseudo-random number generator can satisfy these definitions. Indeed, consider a real number x such as $\sqrt{2}$, π or e which has the property that it is possible to compute the successive binary digits of its base-two expansion. Such x satisfy

$$I(x_n) = I(n) + O(1) = O(\log n),$$

and are therefore maximally non-random. Nevertheless, most real numbers are random. In fact, if each bit of an infinite binary sequence is produced by an independent toss of an unbiased coin, then the probability that it will satisfy (5) is one. We shall now consider a particularly interesting random real number, Ω, discovered by Chaitin [13, p. 336].

A. M. Turing's theorem that the halting problem is unsolvable is a fundamental result of the theory of algorithms [4]. Turing's theorem states that there is no mechanical procedure for deciding whether or not an arbitrary program p eventually comes to halt when run on the universal computer U. Let Ω be the probability that the standard computer U eventually halts if each bit of its program p is produced by an independent toss of an unbiased coin. The unsolvability of the halting problem is intimately connected to the fact that the halting probability Ω is a random real number, i.e., its base-two expansion is a random infinite binary sequence in the very strong sense (5) defined above. From (5) it follows that Ω is normal (a notion due to É. Borel [18]), that Ω is a kollectiv with respect to all computable place selection rules (a concept due to R. von Mises and A. Church [19]), and it also follows that Ω satisfies all computable statistical tests of randomness (this notion being due to P. Martin-Löf [20]). An essay by C. H. Bennett on other remarkable properties of Ω, including its immunity to computable gambling schemes, is contained in [3].

K. Gödel established his famous incompleteness theorem by modifying the paradox of the liar: instead of "This statement is false" he considers "This statement is unprovable." The latter statement is true if and only if it is unprovable; it follows that not all true statements are theorems and thus that any formalization of mathematical logic is incomplete [5-7]. More relevant to algorithmic information theory is the paradox of "the smallest positive integer which cannot be specified in less than a billion words." The contradiction is that the phrase in quotes only has fourteen words even though at least a billion should be necessary. This is a version of the Berry paradox, first published by B. Russell [6, p. 153]. To obtain a theorem rather than a contradiction, one considers instead "the binary string s which has the shortest proof that its complexity $I(s)$ is greater than a billion." The point is that this string s cannot exist. This leads one to the metatheorem that although most bit strings are random and have information content approximately equal to their lengths, it is impossible to prove that a specific string has information content greater than n unless one is using at least n bits of axioms. See [4] for a more complete exposition of this information-theoretic version of

Gödel's incompleteness theorem, which was first presented in [21]. It can also be shown that n bits of assumptions or postulates are needed to be able to determine the first n bits of the base-two expansion of the real number Ω.

Finally, it should be pointed out that these concepts are potentially relevant to biology. The algorithmic approach is closer to the intuitive notion of the information content of a biological organism than is the classical ensemble viewpoint, for the role of a computer program and of DNA are roughly analogous. [22] discusses possible applications of the concept of mutual algorithmic information to theoretical biology; it is suggested that a living organism might be defined as a highly correlated region, one whose parts have high mutual information. See also [23].

REFERENCES

General References

[1] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication,* University of Illinois Press, Urbana, 1949. (The first and still one of the very best books on classical information theory.)

[2] G. J. Chaitin, "Randomness and mathematical proof," *Scientific American* **232**, No. 5 (May 1975), pp. 47-52. (An introduction to algorithmic information theory emphasizing the meaning of the basic concepts.)

[3] M. Gardner, "The random number Ω bids fair to hold the mysteries of the universe," Mathematical Games Dept., *Scientific American* **241**, No. 5 (Nov. 1979), pp. 20-34. (An introduction to algorithmic information theory emphasizing the fundamental role played by Ω.)

[4] M. Davis, "What is a computation?" in *Mathematics Today: Twelve Informal Essays,* L. A. Steen (ed.), Springer-Verlag, New York, 1978, pp. 241-267. (An introduction to algorithmic information theory largely devoted to a detailed presentation of the relevant background in computability theory and mathematical logic.)

[5] D. R. Hofstadter, *Gödel, Escher, Bach: an Eternal Golden Braid,* Basic Books, New York, 1979. (The longest and most lucid introduction to computability theory and mathematical logic.)

[6] J. van Heijenoort (ed.), *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931,* Harvard University Press, Cambridge, 1977. (This book and the next together comprise a stimulating collection of all the classic papers on computability theory and mathematical logic.)

[7] M. Davis (ed.), *The Undecidable – Basic Papers on Undecidable Propositions, Unsolvable Problems And Computable Functions,* Raven Press, Hewlett, 1965.

[8] G. J. Chaitin, "Algorithmic information theory," *IBM Journal of Research and Development* **21** (1977), pp. 350-359, p. 496. (A survey of algorithmic information theory.)

[9] T. L. Fine, *Theories of Probability: An Examination of Foundations,* Academic Press, New York, 1973. (A survey of the remarkably diverse proposals which have been made for formulating probability mathematically. Caution: the material on algorithmic information theory contains some inaccuracies, and it is also somewhat dated due to recent rapid progress in this field.)

Technical References

[10] R. J. Solomonoff, "A formal theory of inductive inference," *Information and Control* **7** (1964), pp. 1-22, pp. 224-254.

[11] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Problems of Information Transmission* **1** (1965), pp. 1-7.

[12] G. J. Chaitin, "On the length of programs for computing finite binary sequences," *Journal of the ACM* **13** (1966), pp. 547-569; **16** (1969), pp. 145-159.

[13] —, "A theory of program size formally identical to information theory," *Journal of the ACM* **22** (1975), pp. 329-340.
[14] L. A. Levin, "Laws of information conservation (nongrowth) and aspects of the foundation of probability theory," *Problems of Information Transmission* **10** (1974), pp. 206-210.
[15] P. Gač, "On the symmetry of algorithmic information," *Soviet Mathematics – Doklady* **15** (1974), pp. 1477-1480.
[16] G. J. Chaitin and J. T. Schwartz, "A note on Monte Carlo primality tests and algorithmic information theory," *Communications on Pure and Applied Mathematics* **31** (1978), pp. 521-527.
[17] H. Feistel, "Cryptography and computer privacy," *Scientific American* **228**, No. 5 (May 1973), pp. 15-23.
[18] M. Kac, *Statistical Independence in Probability, Analysis and Number Theory,* Mathematical Association of America, 1959.
[19] A. Church, "On the concept of a random sequence," *Bulletin of the AMS* **46** (1940), pp. 130-135.
[20] P. Martin-Löf, "The definition of random sequences," *Information and Control* **9** (1966), pp. 602-619.
[21] G. J. Chaitin, "Information-theoretic computational complexity," *IEEE Transactions on Information Theory* **IT-20** (1974), pp. 10-15.
[22] —, "Toward a mathematical definition of 'life'," in *The Maximum Entropy Formalism,* R. D. Levine and M. Tribus (eds.), MIT Press, Cambridge, 1979, pp. 477-498.
[23] F. Papentin, "Complexity of snowflakes," *Naturwissenschaften* **67** (1980), pp. 174-177.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

SOBRE LAS CONDICIONES "ON" DE PL/I Y SU PARAMETRIZACION

Guido Vassallo
Departamento de Computación
Facultad de Ingeniería, Universidad de Buenos Aires

RESUMEN

Se define un fragmento de PL/I, que incluye las construcciones más comunes (asignaciones, bloques, entrada/salida, etc.) y los comandos ON, REVERT, y SIGNAL. Las condiciones consideradas incluyen (algunas de) las predeclaradas y las declaradas por el programador. Se toman en cuenta tanto las acciones establecidas por programa como las del sistema. Se consideran los mecanismos de habilitación e inhabilitación.

Se extiende el minilenguaje con el agregado de parámetros y argumentos para las condiciones, discutiéndose varios modos de pasaje (por valor, por referencia y por valor y resultado), y su especificación tanto en la frase ON como en la SIGNAL.

Finalmente, se sugieren otras posibilidades.

Se utiliza para la definición del significado de las distintas construcciones una versión de la Semántica Matemática de Strachey y Scott.

1. INTRODUCCION

1.1. ANTECEDENTES

Entre las tantas construcciones contenidas en PL/I se destaca el manejo de condiciones.

En general, estas condiciones se asocian a interrupciones originadas durante el proceso, tales como "overflow", fin de archivo, etc. Es posible programar las acciones a efectuar ante cada interrupción, lo que hace más flexible la programación.

Sin embargo, los aspectos lingüísticos esenciales del manejo de condiciones pueden disociarse de los conceptos relativos a interrupciones, utilizando identificadores con el atributo CONDITION - cuyo uso generalizado juega un papel importante en programación concurrente [SCP]. El autor ha encontrado didácticamente oportuno este enfoque, en los cursos regulares dictados en la Facultad de Ingeniería de la Universidad de Buenos Aires.

La descripción precisa de estos mecanismos no resulta fácil, dada la complejidad del lenguaje - y de sus manuales [PL/I]. Consultas casuales hechas a programadores expertos en PL/I han mostrado que la comprensión de ciertos aspectos no es simple, por esa vía.

Una descripción relativamente informal, pero clara y concisa, de los mecanismos mencionados se encuentra en [Wegner] . Ahí figura también el planteo de la posibilidad de extender significativamente la potencia de esas construcciones, mediante su parametrización.

Por otra parte, el Laboratorio de IBM en Viena, abocado a la definición formal de PL/I, inspirándose en ideas de [McCarthy] desarrolló una metodología especial: Vienna Definition Language (VDL) [Lucas y otros] . Este enfoque corresponde a una semántica operacional, en la cual las construcciones lingüísticas son definidas mediante su interpretación en un procesador abstracto.

Simultáneamente, un grupo de investigadores de Oxford, encabezados por Strachey y Scott, han ido desarrollando la semántica matemática [Strachey 66, 73, Milne y Strachey] . La semántica matemática puede ubicarse entre la operacional - de la cual difiere por el menor énfasis dado a cuestiones de tipo implementación - y la semántica axiomática [Hoare 69, 71] , menos constructiva.

La semántica matemática ha resultado particularmente adecuada, y de hecho ha "convertido" al grupo de Viena, que ha desarrollado su propia variante [Bjørner y Jones] . Entre otros resultados, produjo una interpretación del manejo de condiciones parametrizadas, como en la propuesta de [Wegner] .

1.2. PLAN DE TRABAJO

Con el mismo método de un trabajo anterior [Vassallo] , se intenta aquí definir el significado de una familia de mini-sublenguajes de PL/I (con algunas licencias poéticas).

El minilenguaje básico (PL.1), que se describe en 3, incluye, además de las declaraciones de variables y los comandos más comunes, la declaración de identificadores de condición, el establecimiento de condiciones (ON), los comandos SIGNAL y REVERT, las acciones "standard" del sistema, y la habilitación e inhabilitación de condiciones.

Ya saliendo de PL/I, se amplía PL.1 introduciendo varias formas de parametrización para las frases ON y SIGNAL (PL.21 a PL.26).

Se utiliza una variante relativamente primitiva de la semántica matemática, puesto que la omisión de rótulos y transferencias de control en los minilenguajes permite obviar el recurso al concepto de continuación [Strachey y Wadsworth, Milne y Strachey] y expresar el significado de cada comando simplemente como una transición de estados.

En la notación se ha sacrificado algo de concisión a favor de una mejor legibilidad.

2. NOTACIONES BASICAS

No distinguiremos entre tuplas, listas y sucesiones finitas.

Si $x.1,...x.n \in A$, entonces $\langle x.1,...x.n\rangle \in A^n$. Además, $x \neq \langle x\rangle$, es decir $A \neq A^1$.

Por definición la 0-pla o lista nula, nil $= \langle\rangle$, es el único elemento de A°, para cualquier A.

Se usarán $A^+ = A^1 \cup A^2_+ \cup \ldots$
y $A^* = A^\circ \cup A^+$

Se representará por $A \rightarrow B$ el conjunto de todas las funciones parciales con dominio incluído en A y codominio incluído en B. En rigor, en semántica matemática sólo se consideran funciones "buenas" en un cierto sentido.

Al aplicar funciones, a menudo se omitirán paréntesis redundantes; $\ulcorner fx \urcorner$ equivale a $\ulcorner f(x) \urcorner$, y $\ulcorner fhx \urcorner$ a $\ulcorner (f(h))(x) \urcorner$.

"Lo" indefinido se representará por "?". Es, por ejemplo, el "valor" obtenido aplicando una función parcial a un elemento fuera de su dominio. La función totalmente indefinida (vacía) se representará por "$\emptyset$". Las funciones que se usarán serán casi siempre estrictas: f (?)=?.

Se usará un operador dom que aplicado a una función produce una permutación de los elementos de su dominio. Ergo, para cualquier A, B, dom $\in (A \rightarrow B) \rightarrow A^*$.

Si $f \in A \rightarrow B$, puede generalizarse a $A^* \rightarrow B^*$ definiendo
$f(\langle x.1,...x.n\rangle) = \langle f(x.1),...f(x.n)\rangle$
En particular, f(nil) = nil.

Un operador que se utilizará continuamente, para extender y/o modificar funciones es [Reynolds] :

$\underline{ext} \in (A \to B) \times A^n \times B^n \to (A \to B)$ (para cualquier A,B,n).
En general $\underline{ext}$ $(f, \langle x.1,\ldots x.n\rangle , \langle y.1,\ldots y.n\rangle) = g$, donde

$$g(t) = \begin{cases} y.i \text{ si } t=x.i \text{ para algún } i,\ 1 \leq i \leq n \\ f(t) \text{ si no} \end{cases}$$

Todas las x.i deben ser distintas. Se escribirá $\underline{ext}$ (f,x,y) por $\underline{ext}$ (f, ⟨x⟩ , ⟨y⟩).

Se usará la construcción metalingüística

$\underline{if}$ p $\underline{then}$ u $\underline{else}$ v $\underline{fi}$

tal que: si p es verdadero, vale lo mismo que u; si p es falso, vale lo mismo que v; si p es indefinido, resulta indefinida (aunque esto último podría discutirse, si u y v valen lo mismo). Se entiende que, por ejemplo, $\underline{if}$ 0=0 $\underline{then}$ 0-0 $\underline{else}$ 0/0 $\underline{fi}$ está definido (y vale 0) aunque 0/0 no lo está.

La conjunción lógica se define por p & q = $\underline{if}$ p $\underline{then}$ q $\underline{else}$ $\underline{fa}$ $\underline{fi}$; ergo ? & $\underline{fa}$ = ? , $\underline{fa}$ & ? = $\underline{fa}$, siendo $\underline{ve}$ = verdadero, $\underline{fa}$ = falso.

Se aplicarán a listas las funciones:

cabeza	= $\underline{hd} \in A^+ \to A$	(primer elemento de lista no nula)
cola	= $\underline{tl} \in A^+ \to A^*$	(sublista obtenida al decapitar otra)
constructor	= $\underline{cons} \in AxA^* \to A^+$	(lista obtenida yuxtaponiendo un elemento de A y otra lista).

En general, se verifica:

$\underline{hd}$ ($\underline{cons}$ (x,y)) = x
$\underline{tl}$ ($\underline{cons}$ (x,y)) = y
$z \neq \underline{nil} \Rightarrow z = \underline{cons}(\underline{hd}\ z, \underline{tl}\ z)$

Nótese que $\underline{cons}$ no es estricta: $\underline{cons}$ (?, ⟨...⟩) = ⟨?,...⟩ ≠ ?.

Otra función es la longitud de una lista, $\underline{lg} \in A \to \mathbb{N}$, que puede definirse por

$\underline{lg}$ z = $\underline{if}$ z=$\underline{nil}$ $\underline{then}$ 0 $\underline{else}$ 1+$\underline{lg}$($\underline{tl}$ z) $\underline{fi}$.

En algunos casos, se aplicarán índices, o selectores de componentes de tuplas, escritos como sufijos, Así, ⌜x.k⌝ significa x_k, y ⌜z.$\underline{hd}$⌝ equivale a ⌜$\underline{hd}$ z⌝.

3. MINILENGUAJE BASICO (PL.1)

Definiremos un minilenguaje, al cual para futuras referencias denominaremos PL.1.

Este sublenguaje de PL/I incluye:

- algunas de las condiciones, a saber ENDFILE (implícitamente aplicada a SYSIN), CHECK (siempre explícitamente aplicado a una variable), ZERODIVIDE y CONDITION;

- acciones standard del sistema y opción SYSTEM;

- habilitación e inhabilitación de condiciones;

- comando ON, REVERT y SIGNAL; y

- las construcciones ordinarias de cualquier lenguaje imperativo.

3.1. SINTAXIS

Existen las siguientes categorías sintácticas:

Prog	- programas
Cmd	- comandos
Expr	- expresiones
Idf	- identificadores
Cond	- condiciones

Los elementos genéricos de estas categorías se representarán respectivamente por prog, cmd, expr, idf, cond, posiblemente adornados con índices, ápices, etc.

Las producciones de la gramática se expresarán en un BNF informal. Las eventuales ambigüedades e imprecisiones se consideran irrelevantes.

```
prog :: = idf: PROC OPTIONS (MAIN);
          DCL (idf.1,...idf,p)COND,(idf.p+1,...idf.q)VAR;
          cmd.1 ... cmd.n
          END;

cmd  :: = idf = expr ;
        | GET (idf) ;
        | PUT (expr) ;
        | IF expr THEN cmd
        | IF expr THEN cmd.1 ELSE cmd.2
        | DO ; cmd.1 ... cmd.n END ;
        | DO WHILE expr ; cmd.1 ... cmd.n END ;
        | BEGIN ; DCL (idf.1,...idf.h) VAR; cmd.1 ... cmd.n END ;
        | ON cond cmd
        | ON cond SYSTEM;
        | REVERT cond ;
        | SIGNAL cond ;
        | (cond): cmd
        | (NOcond): cmd

cond :: = COND (idf)
        | ENDFILE
        | CHECK (idf)
        | ZDIV
```

```
expr :: = idf
        | expr.1 / expr.2
        | ‹etc›
```

idf a piacere.

Se ha colocado VAR donde correspondería algún otro atributo (p.ej. FLOAT). En ⌜ON cond cmd⌝, no se prohibe, como en PL/I, que cmd sea un grupo DO u otro establecimiento de condición. Tampoco se ha restringido el uso de prefijos a CHECK y ZDIV. Estas restricciones podrían definirse en el nivel sintáctico, distinguiendo comandos simples, bloques y comandos compuestos (IF, DO, ON). Ello complicaría el azúcar sintáctico sin añadir especias semánticas.

Algunas de las restricciones del lenguaje, respecto de PL/I, tales las de no poder omitir o disponer en otro orden las declaraciones, son puramente sintácticas y no disminuyen su potencia. Un simple preprocesamiento permitiría reducir cualquier programa que haga uso de esas licencias a la forma "canónica" aquí definida. En particular, el agrupamiento de las declaraciones de identificadores de CONDición al principio del programa se justifica por ser EXTERNALs.

Una mayor pureza (y menor redundancia) se obtendría usando, en lugar de la sintaxis concreta enunciada, una sintaxis abstracta como las usadas en [McCarthy 66] y [Lucas y otros]. Sin embargo, la presentación resultaría más pesada y no más precisa.

3.2. DOMINIOS Y FUNCIONES SEMANTICAS

3.2.1. VALORES Y DENOTACIONES

Los dominios fundamentales (no ulteriormente analizados) son:

Val: valores "atómicos": lógicos, aritméticos, etc.

Den: denotaciones (que podrán implementarse como ubicaciones de memoria).

Se definirán en base a éstos los demás dominios semánticos.

3.2.2. AMBIENTES

Se define un conjunto de cuaternas denominadas ambientes ("environments"):

$Amb = (Idf \rightarrow Den) \times (Cond \rightarrow Den) \times (Cond \rightarrow Den) \times (Cond \rightarrow \mathbb{B})$

donde $\mathbb{B} = \{ve, fa\}$ es el conjunto de los valores lógicos.

Un elemento genérico de Amb, se escribirá amb o bien ‹amb. var, amb. act, amb. lat, amb. hab›.

Se dirá que si x es un identificador de VARiable, entonces amb.var(x) es su denotación. Si x es un identificador de CONDición, entonces amb. act(x) y amb.lat(x) son respectivamente sus denotaciones actual y latente, y amb.hab(x) su indicador de habilitación.

Según se verá en las ecuaciones semánticas (3.3), el de ambiente es un concepto estático: en cada punto del texto de un programa es válido un ambiente bien determinado.

3.2.3. ESTADOS

Se define un conjunto de cuaternas denominadas estados:

$$Est = Val \times (Den \rightarrow (Val \cup Cla)) \times Val^* \times Val^*$$

Un elemento genérico de Est, se escribirá est o bien
$\langle$ est.val,est.mem,est.ent,est.sal $\rangle$

Según se verá en las ecuaciones semánticas, el de estado es un concepto dinámico: en cada instante del proceso de ejecución de un programa existe un estado bien determinado.

La interpretación intuitiva de (los componentes de) un estado es la siguiente: est.val es el último valor calculado; est.mem es la función memoria, tal que est.mem(x) es el valor o clausura (v. 3.2.4) "contenido" en x; est.ent y est.sal son respectivamente las corrientes de datos de entrada y salida, concebidas como sucesiones de valores.

3.2.4. CLAUSURAS Y ACCIONES DEL SISTEMA

Se define un conjunto de funciones denominadas clausuras:

$$Cla = Est \rightarrow Est$$

Las clausuras corresponden a transiciones de estado "almacenables".

Comparando las definiciones de Est y Cla se advierte una paradoja, puesto que la cardinalidad de cada uno debería superar a la del otro. Este problema está resuelto en semántica matemática, definiendo cuáles funciones son "buenas".

Las acciones standard del sistema son ciertas clausuras fijas, definidas por una función sis $\in$ Cond $\rightarrow$ Cla. Por ejemplo, podría ser que para todo e $\in$ Est, resulte sis (CHECK (idf)) (e) = $\langle$ e.val, e.mem,e.ent, cons (e.val,cons (idf,e.sal))$\rangle$

En PL/I, las acciones standard pueden incluir terminaciones del proceso, no expresables en esta versión de semántica matemática, aunque sí en la que usa continuaciones [Strachey y Wadsworth].

3.2.5. INTERPRETACION DE PROGRAMAS, COMANDOS Y EXPRESIONES

El significado de un programa, comando o expresión se obtendrá a través de la función $:

$$\$ \in (Prog \cup ((Cmd \cup Expr) \times Amb)) \rightarrow (Est \rightarrow Est)$$

Un programa significa una transición, de un estado "inicial" a otro "final". Lo que interesará normalmente serán el componente est.ent

del estado inicial (datos ingresados al programa) y al componente est.sal del estado final (resultados emitidos por el programa).

Un comando, en general, produce una transformación en la memoria, la entrada y la salida. Una expresión "matemáticamente pura" produce un valor. Una expresión con efectos colaterales, como las que habrá que analizar, produce simultáneamente un valor y una transformación. Por eso se conviene en asimilar los procesos de ejecución de un comando y de evaluación de una expresión: ambos producen un nuevo estado, incluyendo el valor calculado (est.val) y los componentes memoria, entrada y salida. De acuerdo al tipo de $, la transformación de estado denotada por un comando o expresión, depende también del ambiente asociado al punto del programa en que figura dicha frase.

3.2.6. GENERACION DE NUEVAS DENOTACIONES

Se usará además una función generatriz de nuevas denotaciones:

$$\underline{nue} \in \mathbb{N} \to Den^*$$

que definiremos informalmente: para todo natural k, nue k es una k-pla de denotaciones "libres".

3.3. ECUACIONES SEMANTICAS

3.3.1. PROGRAMA

$$\$(\ \ulcorner \text{idf: PROC OPTIONS(MAIN);}\ \text{DCL(idf.1,...idf.p)COND,(idf.p+1,...idf.q)VAR;}\ \text{cmd.1 ... cmd.n}\ \text{END;}\urcorner\ )\ (est) =$$

$$= \$(\ \ulcorner \text{BEGIN; DCL(idf.p+1,...idf.q)VAR;cmd.1 ... cmd.n END ;}\urcorner$$
$$, \langle \emptyset\ ,\ \underline{ext}(\emptyset,c,d)\ ,\ \emptyset\ ,\ \underline{ext}(\emptyset,c,\ \langle \underline{ve},...\underline{ve},\underline{ve},\underline{ve}\ \rangle)\rangle$$
$$(\ \langle \text{est.val},\ \underline{ext}\ (\text{est.mem},d,\text{sis}(c)),\ \text{est.ent},\text{est.sal}\rangle\ )$$

donde $c = \langle\ \text{COND(idf.1)},...\text{COND(idf.p)},\text{ENDFILE},\text{ZDIV}\ \rangle$

$d = \underline{nue}\ (p+2)$

El significado de un programa es el mismo que el de un bloque en un ambiente que asocie a cada condición declarada (COND) o predeclarada (ENDFILE, ZDIV) una denotación actual y un indicador de habilitación, y en un estado que se obtiene del inicial inicializando cada denotación de ésas con la correspondiente acción standard del sistema.

3.3.2. ASIGNACION

$$\$(\ \ulcorner \text{idf = expr ;}\urcorner\ ,\ amb)\ (est) =$$

$$= \underline{if}\ \ amb.hab(CHECK(idf))\ \underline{then}\ est".\ mem(amb.act(CHECK(idf)))(est")$$
$$\underline{else}\ est"\ \underline{fi}$$

donde est" = ⟨est'.val,ext(est'.mem,amb.var(idf),est'.val),est'.ent,
est'.sal⟩

est' = $(expr,amb) (est)

La interpretación de una asignación se inicia con la evaluación de su parte derecha, produciendo est'; luego se asocian en memoria la denotación de su parte izquierda y el valor calculado, produciendo est"; finalmente, si la condición CHECK está habilitada para tal parte izquierda, se ejecuta la acción asociada en ese momento a dicha condición.

El valor resultante de la asignación (est".val,si el CHECK no produce efectos extraños) es el mismo que fue transmitido. Esto sugiere inmediatamente la ecuación semántica correspondiente a una asignación múltiple

⌜idf.1,...idf.k = expr;⌝

donde en definitiva se estaría considerando la asignación como una expresión, como en [APL], o [Algol 68] .

3.3.3 LECTURA

$ (⌜GET (idf) ;⌝, amb) (est) =

= if est.ent=nil then est.mem(amb.act(ENDFILE))(est)
else if amb.hab(CHECK(idf))
then est'.mem(amb.act(CHECK(idf)))(est')
else est' fi fi

donde est' = ⟨v,ext(est.mem,amb.var(idf),v),tl(est.ent),est.sal⟩
v = hd(est.ent)

Si la corriente de datos no está agotada, idf pasa a poseer el valor del primer valor de dicha corriente, y ésta pierde la cabeza (el primer valor de est.ent es en cada momento el próximo valor a leer); finalmente, si CHECK está habilitado para idf, se ejecuta la acción correspondiente. Si la corriente de entrada está agotada, se ejecuta la acción asociada a ENDFILE.

3.3.4. EMISION

$ (⌜PUT (expr) ;⌝ , amb) (est) =

= ⟨est'.val,est'.mem,est'.ent,cons (est'.val,est'.sal)⟩

donde est' = $ (expr,amb) (est)

Se evalúa expr y se agrega a la cabeza de la lista de salida el valor calculado. En cada instante hd(est.sal) es el último valor emitido. El valor resultante del comando PUT es el valor emitido.

3.3.5. COMANDOS ALTERNATIVOS

$ (⌜IF expr THEN cmd⌝ , amb) (est) =
= if est'. val then $(cmd,amb)(est') else est' fi

$ (⌜IF expr THEN cmd. 1 ELSE cmd. 2⌝ , amb) (est) =
= if est'.val then $(cmd.1,amb)(est') else $(cmd.2,amb)(est') fi

donde est' = $(expr,amb)(est).

Se evalúa la expresión y se ejecuta luego el comando seleccionado según el valor hallado.

3.3.6. COMANDO COMPUESTO

$ (⌜DO; cmd.1 ... cmd.n END;⌝ , amb) (est) =
= $(cmd.n,amb)(...$(cmd.1,amb)(est)...)
o de otra forma,

$ (⌜DO; cmd.1 ... cmd.n END;⌝ , amb)
= $(cmd.n,amb)° ...° $(cmd.1,amb)

El efecto es el de componer las transiciones de estado correspondientes a cmd.1,...cmd.n. En particular (si n=0):

$(⌜DO; END;⌝ , amb)(est)=est

3.3.7. COMANDO ITERATIVO

$ (⌜DO WHILE expr; cmd.1 ... cmd.n END;⌝ , amb) = W
donde W ∈ Est → Est es tal que para todo est ∈ Est,
W(est)=if est'.val then W($(⌜DO;cmd.1...cmd.nEND;⌝ ,amb)(est'))
else est' fi
donde est' = $(expr,amb)(est).

La ecuación semántica refleja la equivalencia entre

⌜DO WHILE expr; ---⌝ y

⌜IF expr THEN DO; DO; --- DO WHILE expr; --- END;⌝

Se demuestra en semántica matemática que tal ecuación semántica recursiva tiene efectivamente una solución única.

3.3.8. BLOQUE

$ (⌜BEGIN;DCL(idf.1,...idf.h)VAR; cmd.1 ... cmd.n END;⌝ ,amb)
(est) =
= $ (⌜DO; cmd.1 ... cmd.n END;⌝
,< ext(amb.var, < idf.1,...idf.h > , nue h)
,ext(ext(∅,x,d),z,b),amb.act
,ext(amb.hab,z, < fa,...fa >) >)
(< est.val
,ext(ext(est.mem,d,est.mem(amb.act(x))),b,sis(z))
,est.ent , est.sal >)

donde x = dom(amb.act)
d = nue(lg x)
z = ⟨ CHECK(idf.1),...CHECK(idf.h) ⟩
b = nue h

Se ejecuta el comando compuesto constituído por cmd.1,... cmd.n, en un nuevo ambiente (local) y en un nuevo estado. El ambiente local se obtiene del ambiente global amb mediante: la creación de denotaciones para las variables locales; la creación de un ambiente actual local "isomorfo" al global y extendido con denotaciones para las condiciones CHECK aplicadas a las variables locales; la adopción del ambiente actual global como ambiente latente local; y la extensión del ambiente de habilitación global teniendo en cuenta que las condiciones CHECK están inhabilitadas mientras no se las habilite expresamente. En el nuevo estado, a las denotaciones actuales creadas para las condiciones heredadas del bloque externo y para las CHECK de las variables locales, se asocian las clausuras heredadas y las de sistema, respectivamente.

3.3.9. ESTABLECIMIENTO DE CONDICION

$ (⌜ON cond cmd⌝ , amb) (est) =
= ⟨ est.val,ext(est.mem,amb.act(cond),$(cmd,amb)),est.ent,est.sal ⟩

El efecto es comparable al de una asignación en la cual la denotación actual de cond se asocia con la clausura consistente en el significado de cmd.En PL/I se toma $(⌜BEGIN;cmd END;⌝,amb), algo restrictivo.

Esta clausura incluye amb, lo que implica que al ejecutarse, por ej., ⌜SIGNAL cond;⌝ , las eventuales variables que figuren en cmd se interpretarán en amb (el ambiente del establecimiento de condición), y no en el ambiente de la frase SIGNAL. En cambio, el estado al que se aplicará la clausura será el estado en el momento del señalamiento de la condición.

Se notará la similitud con una declaración de procedimiento: se trataría de una "declaración dinámica" [Wegner].

$ (⌜ON cond SYSTEM ;⌝ , amb) (est) =
= ⟨ est.val,ext(est.mem,amb.act(cond),sis(cond)) ,est.ent,est.sal ⟩

En este caso, la clausura asociada es la acción del sistema.

3.3.10. REVERSION DE CONDICION

$ (⌜REVERT cond ;⌝ , amb) (est) =
= ⟨est.val,ext(est.mem,amb.act(cond),est.mem(amb.lat(cond)))
,est.ent,est.sal ⟩

El efecto es comparable al de una asignación, en la cual a la denotación actual de cond se le asocia la misma clausura que a la denotación latente.

En la ejecución de 2 o más REVERTs referidos a la misma condición, sin que entre ellos se ejecuten ON o activaciones/desactivaciones de bloques, sólo el primer REVERT puede tener un efecto no nulo. Al activarse un bloque todo ocurre como si se ejecutara un REVERT generalizado a todas las condiciones declaradas.

3.3.11. SEÑALAMIENTO DE CONDICION

$ (⌜SIGNAL cond;⌝ ,amb) (est) =
=if amb.hab(cond)then est.mem(amb.act(cond))(est) else est fi

El efecto, si la condición está habilitada, es aplicar la clausura asociada a la denotación actual de cond. Nótese la analogía con una invocación de rutina.

3.3.12. COMANDOS PREFIJADOS

$ (⌜(cond):cmd⌝ , amb)
=$(cmd, < amb.var,amb.act,amb.lat,ext(amb.hab,cond,ve) >)

$ (⌜(NOcond):cmd⌝ ,amb)
=$(cmd, < amb.var,amb.act,amb.lat,ext(amb.hab,cond,fa) >)

Un prefijo ("statement prefix") afecta el indicador de habilitación del correspondiente comando. Tratándose de un componente del ambiente, el efecto del prefijo es válido sólo durante la ejecución del comando al cual se ha prefijado.

No hemos seguido estrictamente las reglas de PL/I, según las cuales en ⌜(cond): IF expr THEN cmd⌝ , por ejemplo, el prefijo valdría en expr pero no en cmd.

3.3.13. VALOR DE UN IDENTIFICADOR

$ (idf , amb) (est) = est.mem(amb.var(idf))

El valor poseído por un identificador es el asociado a (o contenido en) su denotación.

3.3.14. VALOR DE UN COCIENTE

$(⌜expr.1/expr.2⌝ , amb) (est) =
= if amb.hab (ZDIV)
  then if num(est'.val) & est'.val ≠ 0
       then if num(est".val)
            then < est".val/est'.val,est".mem,est".ent,est".sal >
            else est".mem(amb.act(ZDIV)) (est") fi
       else est'.mem(amb.act(ZDIV)) (est') fi
  else < est".val/est'.val,est".mem,est".ent,est".sal > fi
donde est' = $ (expr.2,amb)(est)
     est" = $ (expr.1,amb) (est')
     num ∈ Val → 𝔹 es el predicado característico del subconjunto de valores numéricos.

En PL/I, la verificación aquí realizada con num, se efectúa parcialmente en compilación, y eventualmente por la condición CONVERSION.

Se deduce de la ecuación dada que si ZDIV no está habilitada puede producirse una situación de error no ulteriormente analizada aquí.

3.4. CONSTRUCCIONES ADICIONALES

Se indicarán algunas construcciones no pertenecientes a PL/I, que pueden ser fácilmente definidas en el formalismo utilizado para PL.1.

3.4.1. CONSTRUCCION "AT END"

Se agrega la producción

```
    cmd :: = GET (idf) AT END cmd
```

Su significado se define por

```
    $(⌜GET(idf) AT END cmd⌝ , amb) (est) =
= if est.ent=nil then $(cmd,amb)(est)
  else if amb.hab(CHECK(idf))
       then est'.mem(amb.act(CHECK(idf)))(est')
       else est'   fi   fi

donde est' = ⟨v,ext(est.mem,amb.var(idf),v),tl(est.ent),est.sal⟩
      v    =     hd  (est.ent)
```

Comparando con las ecuaciones anteriores, se advierte que esta construcción

```
no equivale a    ⌜ON ENDFILE cmd GET (idf);⌝
ni a             ⌜ON ENDFILE cmd GET (idf); REVERT ENDFILE;⌝
sino a           ⌜BEGIN; ON ENDFILE cmd GET (idf); END;⌝
```

(Este tipo de discusión se haría más difícil con las usuales definiciones de manual).

3.4.2. CONTROL DINAMICO DE ASERCIONES

En Algol W existe un comando de la forma

assert ⟨expresión lógica⟩ .

Este comando, inspirado en las aserciones de [Floyd], [Hoare 69], etc., difiere de éstas por tratarse de expresiones (del lenguaje de programación) que son verificadas dinámicamente (en ejecución) y no estáticamente. Por lo tanto no sirven para demostrar que un programa es correcto, aunque sí para corroborarlo, como herramienta de depuración.

El significado de este comando puede encuadrarse en lo ya visto, considerando un nuevo tipo de condición, inhabilitada si no se la habilita expresamente, y que se invocaría por una frase ASSERT más bien que con SIGNAL.

Se agregan las producciones:

```
cmd   :: = ASSERT expr;
cond  :: = ASSERTION
```

En la interpretación de un programa (3.3.1.) deberá tomarse en cuenta la inicialización con sis (ASSERTION), y fa.

La única ecuación semántica a agregar sería

```
    $ ( ⌜ASSERT expr ;⌝ , amb) (est) =
= if amb.hab(ASSERTION)
  then if est'.val then est'
       else est'.mem(amb.act(ASSERTION)) (est')  fi
  else est   fi
donde est' = $(expr,amb)(est)
```

Comparando con las ecuaciones anteriores se advierte que esta construcción no equivale en general a

```
⌜IF ¬(expr) THEN SIGNAL ASSERTION;⌝
```

a menos que esté en un contexto en el cual ASSERTION esté habilitada.

3.4.3. EXPRESIONES BLOQUE Y COMANDO RETURN

En [Algol W] , [Algol 68] y [BCPL] , existen expresiones precedidas por bloques, que al estilo PL/I se escribirían

```
⌜DO  ; cmd.1 ... cmd.n-1 RETURN(expr); END;⌝
⌜BEGIN;DCL (idf.1,...idf.h)VAR;cmd.1...cmd.n-1 RETURN(expr);END;⌝
```

Su interpretación estaría dada exactamente por las ecuaciones semánticas 3.3.6 y 3.3.8, definiendo

```
    $ ( ⌜RETURN(expr);⌝  ,amb) = $(expr , amb)
```

3.4.4. EXPRESION VALOR LEIDO

En [BCPL] , p. ej., se tienen expresiones como readnum (), cuyo valor es el del primer número de la corriente de entrada.

Esta construcción puede agregarse a PL.1 con la producción

```
    expr :: = READVAL
```

y la ecuación semántica

```
    $ (READVAL , amb) (est) =
= if est.ent=nil then est.mem(amb.act(ENDFILE))(est)
  else < hd(est.ent),est.mem,tl(est.ent),est.sal>   fi
```

3.5. NOTA SOBRE INVOCACION MULTIPLE Y RECURSIVIDAD

La ejecución del comando asociado a una condición puede incluir el señalamiento de otra. En tal caso se dice [PL/I] que se están ejecutando "unidades ON descendentes". Las ecuaciones semánticas dadas muestran que la última unidad ON activada es la primera en completar su ejecución.

En particular pueden coexistir varias activaciones de una misma unidad ON. Es decir, pueden establecerse condiciones recursivas. Así, el programa

```
Q: PROC OPTIONS (MAN); DCL(C)COND,(V)VAR;
   ON COND(C) BEGIN;
     PUT(V); IF V THEN DO; V = ¬ V; SIGNAL COND (C); END;
     ELSE  PUT ('UF'); END;
   V = '1' B; SIGNAL COND (C);
   END;
```

produciría como salida ⟨"UF", fa, ve ...⟩ .

Puede recordarse que, en PL/I, los procedimientos son recursivos sólo si se especifica tal propiedad, y (en ese caso) la semántica de la correspondiente declaración lo refleja [Vassallo] definiendo circularmente el nuevo ambiente creado. No ocurre lo mismo en el caso de establecimientos de condición, debido a que éstos no son declaraciones que alteren el ambiente, sino comandos que alteran el estado.

4. PARAMETRIZACION DE LAS CONDICIONES (PL.2k)

[Wegner] señala la similitud de las construcciones ON y SIGNAL, respectivamente, con "declaraciones dinámicas" e invocaciones de rutinas sin parámetros, y sugiere su parametrización.

Esto resultaría ventajoso si se considera, por ejemplo, que en muchos casos se desea reaccionar ante la ocurrencia de una condición emitiendo un mensaje variable.

La parametrización puede describirse con los mismos esquemas de la definición semántica de procedimientos ([Milne y Strachey] , [Vassallo] , [Donahue]). En efecto, las clausuras que se utilizan en este trabajo coinciden con las que se usan para evaluar procedimientos en la descripción de lenguajes del tipo de Algol.

Sin embargo, en esos casos, la clausura ("código puro" más lista de variables libres, o equivalente) es mantenida en el ambiente y no en el estado, por tratarse de declaraciones estáticas, de modo que los identificadores de procedimientos mantienen su significado en todo su alcance textual. En cambio, en lenguajes como [BCPL] , en el cual las clausuras son "almacenadas", sería posible asimilar la frase ON a una asignación y la SIGNAL a una invocación de rutina.

En la familia de minilenguajes que se definirán en esta sección, se agregarán parámetros formales en las frases ON y argumentos efectivos en la SIGNAL. Entre los varios modos de pasaje se discutirán tres: por valor, por referencia y por valor/resultado. Se examinará luego la posibilidad de determinar el mecanismo de pasaje en la ON y en la SIGNAL, la inclusión de parámetros de tipo condición y otras cuestiones.

Se definirán solamente las construcciones lingüísticas que en virtud de la parametrización se modifiquen respecto de PL.1.

4.1. PASAJE POR VALOR (PL.21)

El mecanismo por valor ("by value") es el que usa PL/I para transmitir argumentos que no sean identificadores.

4.1.1. SINTAXIS

Se agregan las producciones

```
cmd  :: = ON cond(idf.1,...idf.k) cmd
        | SIGNAL cond(expr.1,...expr.k);
```

Sintácticamente, los parámetros formales (idf.1,...idf.k) de la ON se usarán como identificadores de VARiables en cmd.

4.1.2. DOMINIOS SEMANTICOS

Se modifica la definición de clausura:

$Cla = Val^* \times Est \rightarrow Est$

Una clausura representa una transición de estado dependiente de (0 o más) valores. Podría también haberse definido $Cla = Val^* \rightarrow (Est \rightarrow Est)$.

4.1.3. ECUACIONES SEMANTICAS

$\$(\ulcorner \text{ON cond(idf.1,...idf.k) cmd} \urcorner\ , amb)\ (est) =$
$= \langle est.val, \underline{ext}(est.mem, amb.act(cond), f), est.ent, est.sal \rangle$
donde $f \in Cla$ es tal que para todo $v \in Val^k$, $e \in Est$:
$f(v,e) = \$(cmd, \underline{ext}(amb.var, \langle idf.1,...idf.k \rangle, d), amb.act, amb.lat, amb.hab)$
$(\langle e.val, \underline{ext}(e.mem, d, v), e.ent, e.sal \rangle)$
donde $d = \underline{nue}\ k$.

$\$(\ulcorner \text{SIGNAL cond(expr.1,...expr.k);} \urcorner\ , amb)\ (est) =$
$= \underline{if}\ amb.hab(cond)$
$\underline{then}\ e^k.\ mem(amb.act(cond))(\langle e^1.val, ... e^k.val \rangle, e^k)$
$\underline{else}\ est\ \underline{fi}$
donde $e^i = \$(expr.i, amb)\ (e^{i-1})$
$e^\circ = est$

Al señalarse una condición, se evalúan los argumentos (de izquierda a derecha) y se aplica a los valores y estado resultantes la clausura correspondiente.

4.2. PASAJE POR REFERENCIA (PL.22)

El pasaje por referencia ("by reference", "by location") es el mecanismo usual en PL/I para transmitir argumentos efectivos que sean identificadores, y en Fortran para transmitir arreglos. Permite alterar los valores de los argumentos.

La versión que aquí se presenta tiene la desventaja de admitir únicamente identificadores, como argumentos efectivos. Una versión más general está incluída en PL.25.

4.2.1. SINTAXIS

```
cmd  :: = ON cond (idf.1,...idf.k) cmd
        | SIGNAL cond(idf'.1,...idf'.k);
```

4.2.2. DOMINIOS SEMANTICOS

Cla = Den* x Est → Est

Una clausura representa aquí una transición de estado dependiente de 0 o más denotaciones.

4.2.3. ECUACIONES SEMANTICAS

$ (⌜ON cond(idf.1,...idf.k) cmd⌝ , amb) (est) =
= ⟨est.val,ext(est.mem,amb.act(cond),f),est.ent,est.sal⟩
donde f ∈ Cla es tal que para todo d ∈ Den^k, e ∈ Est, f(d,e) =
=$(cmd, ⟨ext(amb.var, ⟨idf.1,...idf.k⟩ ,d),amb.act,amb.lat,amb.hab⟩) (e)

$ (⌜SIGNAL cond(idf'.1,...idf'.k);⌝ ,amb) (est) =
= if amb.hab(cond)
then est.mem(amb.act(cond))(amb.var ⟨ idf'.1,...idf'.k ⟩,est)
else est fi

Al señalar la condición se ejecutará el comando asociado a la misma en un ambiente,ampliación del del establecimiento, en el que son sinónimos (tienen iguales denotaciones) los parámetros formales y los argumentos efectivos correspondientes

4.3. PASAJE POR VALOR/RESULTADO (PL.23)

El pasaje por valor/resultado ("by value result") es el usual en Fortran para transmitir argumentos correspondientes a parámetros escalares (no es exactamente el mismo que se define en [Algol W]).

Este mecanismo y el pasaje por referencia producen efectos coincidentes en muchos casos, pero no siempre.

En esta versión, como en PL.22,los argumentos deben ser identificadores. Una generalización puede hacerse utilizando denotaciones anónimas, como en PL.25.

4.3.1. SINTAXIS

cmd :: = ON cond (idf.1,...idf.k) cmd
| SIGNAL cond (idf'.1,...idf'.k) ;

4.3.2. DOMINIOS SEMANTICOS

Cla = (Den* x Est) → Est

4.3.3. ECUACIONES SEMANTICAS

$ (⌜ON cond (idf.1,...idf.k) cmd⌝ , amb) (est)=
⟨est.val,ext(est.mem,amb.act(cond),f), est.ent,est.sal⟩
donde f ∈ Cla es tal que para todo d ∈ Den^k , e ∈ Est,
f(d,e) = ⟨ e'.val,ext(e'.mem,d,e'.mem(d')),e'.ent,e'.sal⟩
donde d' = nue k
e' = $ (cmd
, ⟨ ext(amb.var,⟨idf.1,...idf.k⟩ ,d')
,amb.act,amb.lat,amb.hab ⟩)
(⟨ e.val,ext(e.mem,d',e.mem(d)),e.ent,e.sal⟩)

```
    $ ( ⌜SIGNAL cond (idf'.1,...idf'.n);⌝  ,amb) (est) =
= if amb.hab(cond)
  then est.mem(amb.act(cond))(amb.var < idf'.1,...idf'.k > ,est)
  else est     fi
```

Al señalarse la condición se ejecutará el comando asociado a la condición, en un ambiente, ampliación del del establecimiento, en el cual se dota a los parámetros formales con nuevas denotaciones, y en un estado en el cual a dichas denotaciones se asocian los valores poseídos por los argumentos efectivos. Finalmente, se copian "en" las denotaciones de los argumentos efectivos los resultados que quedaron asociados a las nuevas.

4.4. MODO DE PASAJE ESPECIFICADO EN EL ESTABLECIMIENTO DE CONDICION (PL.24)

En algunos casos será preferible la transmisión por valor (que no impone restricciones a la forma de los argumentos efectivos) y en otros la transmisión por referencia o por valor/resultado (que permiten modificar argumentos).

Para disponer de ambas posibilidades se especificará el modo de pasaje al establecer una condición, en la tradición de ⟦ Algol ⟧ . Otra solución se verá como PL.25.

4.4.1. SINTAXIS

```
cmd   :: = ON cond((idf.1,...idf.k)VAL,(idf.k+1,...idf.m)REF)cmd
         | SIGNAL cond(expr.1,...expr.k;idf'.k+1,...idf'.m);
```

También podrían tenerse especificaciones VAL_RES (o RES).

4.4.2. DOMINIOS SEMANTICOS

```
Cla = (Val* x Den* x Est) → Est
```

4.4.3. ECUACIONES SEMANTICAS

```
    $ ( ⌜ON cond((idf.1,...idf.k)VAL,(idf.k+1,...idf.m)REF)cmd⌝
      ,amb)  (est) =
= <est.val,ext(est.mem,amb.act(cond),f),est.ent,est.sal>
donde f ∈ Cla es tal que para todo v ∈ Val^k, d ∈ Den^(m-k), e ∈ Est,
f(v,d,e) = $ (cmd
       , < ext(ext(amb.var, < idf.1,...idf.k > ,d')
             , < idf.k+1,...idf.m > , d)
           ,amb.act,amb.lat,amb.hab > )
       ( <e.val,ext(e.mem,d',v),e.ent,e.sal> )
donde d' = nue k
    $ ( ⌜SIGNAL cond(expr.1,...expr.k;idf'.k+1,...idf'.m);⌝
      , amb)  (est) =
= if amb.hab(cond)
  then e^k.mem(amb.act(cond))
       ( < e^1.val,...e^k.val>  ,amb.var < idf'.k+1,...idf'.m > , e^k)
  else est     fi
donde e^i = $(expr.i,amb) (e^(i-1))
      e^0 = est
```

4.5. MODO DE PASAJE ESPECIFICADO EN EL SEÑALAMIENTO DE CONDICION (PL.25)

El problema planteado en 4.4 puede también resolverse "a la PL/I", reservando la decisión acerca del modo de pasaje a la invocación (SIGNAL) y no al establecimiento (ON) de la condición.

La transmisión será por referencia en el caso de identificadores, por ejemplo en ⌜SIGNAL COND(C)(X);⌝ ; y será equivalente a una transmisión por valor para otros argumentos efectivos, por ejemplo en ⌜SIGNAL COND(C)(4);⌝ .

Se utilizará en todos los casos transmisión por referencia, pero creando, para los argumentos efectivos que no sean identificadores, denotaciones "anónimas" (fuera del dominio de su ambiente).

4.5.1. SINTAXIS

```
cmd  :: = ON cond (idf.1,...idf.k) cmd
        | SIGNAL cond (expr.1,...expr.k);
```

4.5.2. DOMINIOS SEMANTICOS

$$Cla = (Den^* \times Est) \to Est$$

4.5.3. ECUACIONES SEMANTICAS

$$\$(\ulcorner \text{ON cond (idf.1,...idf.k) cmd} \urcorner\ , amb)(est) =$$
$$= \langle est.val, \underline{ext}(est.mem, amb.act(cond), f), est.ent, est.sal \rangle$$

donde $f \in Cla$ es tal que para todo $d \in Den^k$, $e \in Est$,

$$f(d,e) = \$(\text{cmd}, \langle \underline{ext}(amb.var, \langle idf.1,...idf.k \rangle, d), amb.act, amb.lat, amb.hab \rangle)(e)$$

$$\$(\ulcorner \text{SIGNAL cond (expr.1,...expr.k);} \urcorner\ , amb)(est) =$$
$$= \underline{if}\ amb.hab(cond)\ \underline{then}\ est.mem(amb.act(cond))\ (\langle d^1,...d^k \rangle\ , e^k)\ \underline{else}\ est\ \underline{fi}$$

donde
$$d^i = \underline{if}\ expr.i \in Idf\ \underline{then}\ amb.var(expr.i)\ \underline{else}\ nue\ 1\ \underline{fi}$$
$$e^i = \underline{if}\ expr.i \in Idf\ \underline{then}\ e^{i-1}\ \underline{else} \langle e'^i.val, \underline{ext}(e'^i.mem, d^i, e'^i.val), e'^i.ent, e'^i.sal \rangle\ \underline{fi}$$
$$e'^i = \$(expr.i, amb)(e^{i-1})$$
$$e^\circ = est$$

Al señalarse la condición, se ejecuta la clausura asociada a su denotación actual en un ambiente en el cual a cada parámetro formal corresponde una denotación (la del argumento efectivo si éste es un identificador, o una "anónima"), y en un estado en el cual cada una de estas denotaciones contiene el valor del argumento efectivo correspondiente. En particular, si todos los argumentos efectivos son identificadores, entonces dicho estado (e^k) coincide con el del señalamiento (est).

4.6. PARAMETROS DE TIPO CONDICION (PL.26).

Las construcciones anteriores pueden generalizarse, admitiendo que como argumentos a transmitir figuren no sólo expresiones que posean valores en Val, sino también condiciones que posean clausuras en Cla.

Así, podría escribirse, por ejemplo, el programa

```
P : PROC OPTIONS (MAIN) ; DCL (D,E) COND;
    ON COND(D)((U)COND,(W)VAR) DO;SIGNAL U;ON U PUT(W);END;
    ON COND(E) PUT('AH') ;
    SIGNAL COND(D) (COND(E),'OH') ;
    SIGNAL COND(E) ;
    END ;
```

obteniendo sucesivamente la emisión de "AH" y de "OH".

Se ampliará al efecto el lenguaje PL.25, transmitiendo las denotaciones de los argumentos variables, y las denotaciones actuales y latentes e indicadores de habilitación de los argumentos que sean condiciones.

4.6.1. SINTAXIS

cmd :: = ON cond((idf.1,...idf.p)COND,(idf.p+1,...idf.q)VAR)cmd
| SIGNAL cond.0 (cond.1,...cond.p,expr.p+1,...expr.q);

4.6.2. DOMINIOS SEMANTICOS

$Cla = (Den^* \times Den^* \times \mathbb{B}^* \times Den^* \times Est) \rightarrow Est$

4.6.3. ECUACIONES SEMANTICAS

$\$(\ulcorner$ON cond((idf.1,...idf.p)COND,(idf.p+1,...idf.q)VAR)cmd$\urcorner$, amb) (est) =
$= \langle est.val, \underline{ext}(est.mem, amb.act(cond), f), est.ent, est.sal \rangle$
donde $f \in Cla$ es tal que para todo $a, l \in Den^p$, $h \in \mathbb{B}^p$,
$d \in Den^{q-p}$, $e \in Est$,
$f(a,l,h,d,e) =$
$= \$(cmd, \langle \underline{ext}(amb.var, \langle idf.p+1,...idf.q \rangle, d), \underline{ext}(amb.act,c,a), \underline{ext}(amb.lat,c,l), \underline{ext}(amb.hab,c,h) \rangle)(e)$
donde $c = \langle idf.1,...idf.p \rangle$

$\$(\ulcorner$SIGNAL cond.0(cond.1,...cond.p,expr.p+1,...expr.q);$\urcorner$, amb) (est) =
$=$ if amb.hab(cond.0)
then e^{q-p}. mem(amb.act(cond.0))
(amb.act(x),amb.lat(x),amb.hab(x), $\langle d^{p+1},...d^q \rangle$, e^q)
else est fi
donde $x = \langle cond.1,...cond.p \rangle$
d^i = if expr.i $\in$ Idf then amb.var(expr.i) else nue 1 fi
e^i = if expr.i $\in$ Idf then e^{i-1}
else $\langle e'^i.val, \underline{ext}(e'^i.mem, d^i, e'^i.val), e'^i.ent, e'^i.sal \rangle$ fi
$e'^i = \$(expr.i, amb)(e^{i-1})$
$e^p = est$

4.7. NOTA SOBRE PASAJE POR NOMBRE

El mecanismo de transmisión por nombre ("by name"), con su potencia (y su opacidad referencial) puede describirse, siguiendo el informe [Algol], mediante una substitución textual (no trivial) de los parámetros formales por los argumentos efectivos en el comando correspondiente a la condi-

ción. Esto sugeriría "almacenar" no ya la clausura $(cmd,amb) sino el propio texto cmd, y el ambiente amb.

Otra alternativa es el uso de "thunks" (Ingermann): ello implicaría la introducción de clausuras para procedimientos funcionales, que produzcan valores (además de eventuales cambios de estado).

Una de las aplicaciones que darían interés al pasaje por nombre sería la posibilidad de transmitir un comando. Por otra parte, esto puede probablemente simularse en PL.26, transmitiendo una condición establecida con ese comando.

BIBLIOGRAFIA

[Algol] P. Naur (Comp): Revised Report on the Algorithmic Language Algol 60 - reimpreso en [Rosen] .

[Algol W] R. Sites: Algol W. Reference Manual - Stanford Univ., 1972

[Algol 68] C.H. Lindsey, S.G. van der Meulen: Informal Introduction to Algol 68 North-Holland, 1973.

[APL] K. Iverson: A Programming Language - Wiley, 1962.

[BCPL] M. Richards, C. Withby-Strevens: BCPL, the Language and its Compiler - Cambridge Univ. Press, 1979.

[Bjørner y Jones] D.Bjørner, C.B.Jones (Comp.): The Vienna Development Method: The Meta-Language - Springer-Verlag, 1978

[Braffort y Hershberg] P. Braffort, D.Hershberg (Comp.): Computer Programming and Formal Systems - North-Holland, 1963

[Donahue] J.E. Donahue: Complementary Definitions of Programming Language Semantics - Springer-Verlag, 1976.

[Engeler] E.Engeler (Comp.): Symposium on Semantics of Algorithmic Languages- Springer-Verlag, 1971.

[Gries] D.Gries: Compiler Construction for Digital Computers - Wiley, 1971.

[Hoare 69] C.A.R. Hoare: The Axiomatic Basis of Computer Programming - Comm. ACM 12 10, 1969.

[Hoare 71] C.A.R.Hoare: Procedures and Parametres: an Axiomatic Approach - en [Engeler]

[Lucas y otros] Lucas, Lauer,Stigleitner: Method and Notation for the Formal Definition of Programming Languages - IBM TR 25087.

[McCarthy 63] J. McCarthy: A Basis for a Mathematical Theory of Computation - en [Braffort y Hershberg]

[McCarthy 66] J.McCarthy: A Formal Description of a Subset of Algol - en [Steel].

[Milne y Strachey] R.E.Milne, C. Strachey: A Theory of Programming Language Semantics - Chapman and Hall, 1976.

[NCC] :Standard Fortran Programming Manual - National Computing Centre, 1972.

[PL/I] : OS PL/I Checkout and Optimizing Compilers: Language Reference Manual- IBM GC 33-0009-4, 1976.

[Pratt] T.W.Pratt: Programming Languages: Design and Implementation - Prentice-Hall, 1975.

[Reynolds] J.C.Reynolds: Definitional Interpreters for Higher-Order Programming Languages - Proc. ACM 25th Nat.Conf., 1972.

[Rosen] S.Rosen(Comp.): Programming Systems and Languages - Mc.Graw Hill,1967.

[SCP] R.C.Holt,G.S.Graham, E.D.Lazowska, M.A. Scott: Structured Concurrent Programming With Operating Systems Applications - Addison Wesley, 1978.

[Steel] T.B.Steel(Comp.):Formal Language Description Languages - North-Holland 1966.

[Strachey]66 C.Strachey: Towards a Formal Semantics - en [Steel]

[Strachey]73 C.Strachey: The Varieties of Programming Languages - Oxford Univ. Press, 1974.

[Vassallo] G.Vassallo: Mathematical Semantics: a Tool to Survey, Relate and Combine Programming Languages - Univ. of Essex, 1974.

[Wegner] P. Wegner: Programming Languages, Information Structures and Machine Organization - McGraw Hill, 1968.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

ESTRUCTURAS DE DATOS, COMPUTACION DETERMINISTICA E IMPLEMENTACION DE PROLOG

PARTE I: AXIOMATIZACION DE LAS ESTRUCTURAS DE DATOS DE PROLOG

Philippe Roussel
Gerard Battani
Ascánder Suarez
Universidad Simón Bolívar. Caracas-Venezuela

INTRODUCCION:

PROLOG es un lenguaje de muy alto nivel no determinístico basado en la lógica de primer orden, usa los conceptos de resolución y unificación {ROB 65} , fué desarrollado en la universidad de Marseille (Groupe d'Intelligence Artificielle) {COL 72}{ROU 75} basado en ideas desarrolladas conjuntamente con la Universidad de Edimburgh (Department of Artificial Intelligence) {KOW 75} . Desde entonces se ha difundido ampliamente y se ha aplicado a preguntas y respuestas en Francés {COL 78} Castellano {DAH 78} y otro idiomas, manipulaciones algebraicas {BER 73 } generación de planes {WAR 74} demostración de teoremas {COE 75} comprehensión de frases habladas {BAT 75} , interpretación de imágenes, bases de datos lógicos {PIQ 79} aplicaciones químicas, farmacéuticas e industriales {PUT 77} .

- Se van a describir dos tipos de expresiones. Expresiones Externas (EE) y expresiones internas (EI).
- Cada una es una manera de representar términos del lenguaje PROLOG.
- Las EE son las que se usan habitualmente en el lenguaje mientras que las EI son más adaptadas a una implementación del mismo.

I. Las Expresiones Externas

- Se componen de los subconjuntos dos a dos disjuntos:

  A de atómos

  V de variables

  F de funciones

  LNV de listas no vacias.

- Sintacticamente las expresiones externas se definen recursivamente como:

  - Un <u>atómo</u> es una secuencia finita de caracteres

    ejemplo [ ] ó atómo.

  - Una <u>variable</u> es una secuencia finita de caracteres

    ejemplo VARIABLE ó X1.

  - Una <u>función</u> es de la forma $f(e_1,e_2,\ldots, e_n)$

    donde f(el símbolo funcional) es una secuencia finita de caracteres y $e_1,\ldots,e_n$ (con $n \geq 1$) son expresiones externas de longitud finita.

  - Una <u>lista no vacía</u> es de una de las dos formas

    (1) $[e_1,e_2,\ldots,e_n]$

    ó (2) $[e_1,e_2,\ldots,e \mid e_{n+1}]$

    } donde $n \geq 1$ y $e_1,\ldots e_n$, e_{n+1} son expresiones externas de longitud finita

- Se llaman $e_1,\ldots, e_n, e_{n+1}$ elementos de la lista.
- La longitud de una expresión externa es el número de caracteres que la componen.

NOTA
- Generalmente se ponen restricciones a esas definiciones sintácticas ambiguas para diferenciar sintacticamente los elementos de cada tipo.

  Por ejemplo las variables siempre empezaran por una letra mayúscula .
- Se puede notar que $[\ ]$ es un átomo pero no es una lista no vacia. Se dice que $[\ ]$ es la lista vacia y se define el conjunto L de listas como $L = LNV \cup [\ ]$.

  Se dice que una expresión externa esta en forma, normal si es un átomo, una variable, una lista en forma normal, ó una función en forma normal.

Definición:
- Se dice que una lista no vacia ℓ esta en forma normal si ℓ es de una de las dos formas

  (3) $[e_1,\ldots, e_n]$

  (4) $[e_1,\ldots, e_n \mid e_{n+1}]$ donde $n \geq 1$, $e_{n+1} \notin L$

  y $e_1,\ldots e_{n+1}$ son expresiones externas en forma normal.
- Se dice que una función esta en forma normal si sus argumentos son EE en formal normal.

- Se nota LFN el conjunto de listas no vacias en forma normal

  FFN el conjunto de funciones en forma normal

  EEFN el conjunto de expresiones externas en forma normal.
- Se puede notar que $[[\ ]]$ en una lista no vacia en forma normal, su

único elemento es la lista vacia.

Definición: Se define la relación binaria $\Longrightarrow$ (ó regla de reescritura) sobre las LNV con las tres reglas:

(5) $[x_1,\ldots,x_n \mid [y_1,\ldots,y_m]] \Longrightarrow [x_1,\ldots x_n,y_1,\ldots,y_n]$

(6) $[x_1,\ldots,x_n \mid [y_1,\ldots y_m|y_{m+1}]] \Longrightarrow [x_1\ldots.,x_n,y_1,\ldots,y_m|y_{,+1}]$

(7) $[x_1,\ldots,x_n|[\,]] \Longrightarrow [x_1,\ldots,x_n]$

donde $n,m \geq 1$, $x_1,\ldots,x_n,y_1,\ldots,y_m,y_{m+1}$ son EE en forma normal

Definición: Se define $u \Longrightarrow^i v$ con $u,v \in$ LNV

como $u \Longrightarrow^{\circ} v$ ssí $u = v$

para $i > 0$ $u \Longrightarrow^i v$ ssí (a) $\exists$ $z \in$ LNV tal que $u \Longrightarrow z$ y $z \Longrightarrow^{i-1} v$

(b) existe un elemento u_j de u que no esta en forma normal. (es el más a la izquierda, es decir, no existe u_ℓ elemento de u, con $\ell < j$, que no esté en forma normal) entonces (b1) si u_j es una lista : existen z lista y z_k con z_k elemento de Z tales que $u_j \Longrightarrow z_k$

y $z \Longrightarrow^{i-1} v$

(b2) si u_j es una función, existe w_n su primer argumento que no esta en forma normal:

existen z, función y z_k argumento de z tales que

$w_n \Longrightarrow z_k$

y $z \Longrightarrow^{i-1} v$.

Se dice que $u \Rightarrow^{*} y$ si $\exists\ i \geq 0$ tal que $u \Rightarrow^{i} v$.

Ejemplos: $[a|[b|c]] \Rightarrow [a,b|c]$

$[[h|[\,]]\ ,\ [d|[e|g]]\ ,\ f[a|[b|c]] \Rightarrow *[[h]\ ,\ [d|[e|g]]]\ ,\ f[a|[b|c]] \Rightarrow *$

$[[h]\ ,\ [d,e|g]\ ,\ f[a|[b|c]] \Rightarrow *\ [[h]\ ,\ [d,e|g]\ ,\ f[a,b|c]]$;

Se puede notar que si se considera $\Rightarrow$ como una regla de simplificación sobre las listas $\Rightarrow *$ esta definida de manera de imponer un orden de simplificación : primero se simplifican los **elementos** de la lista empezando **por la izquierda** , y solo despues se simplifica la lista.

Propiedad 1. $\Rightarrow *$ es un orden parcial sobre LNV.

Mostremos que es antisimétrica por eso definimos para todo $\ell \in$ LNV la función entera $pc(\ell)$ como el número de pares de corchetes de la lista ℓ.

como
$$\begin{cases} pc([\,]) = 1 \\ pc(e) = 0 \quad \text{si } e, \text{ es un átomo distinto de } [\,] \text{ ó una variable.} \\ pc(f(e_1,\ldots e_n)) = \sum_{i=1}^{n} pc(ei) \quad \text{con } e_i \text{ expresiones externas.} \\ pc([e_1,\ldots e_n]) = 1 + \sum_{i=1}^{n} pc(e_i) \text{ con } e_i \text{ expresiones externas.} \end{cases}$$

verificamos que $\forall\ u,v \mid$ tales que $\exists i$ tal que $u \Rightarrow^{i} v$ entonces

$$pc(u) < pc(v) \quad \text{(por inducción)}$$

entonces si $\exists\ i > 0$ tal que $u \Rightarrow^{i} v$ no existe j t_q $v \Rightarrow^{i} u$.

Propiedad 2: z ε LNV es un infimo de $\Longrightarrow^*$ sí z está en forma normal.

Prueba ← Si z está en forma normal no se aplica (5), (6) ó (7) sobre z ni tampoco sobre sus elementos puesto que estan también en forma normal, por lo tanto no existe z' tal que $z \Longrightarrow^* z'$ $(z \neq z')$ y z es un infimo para $\Longrightarrow^*$

→ Si z es un infimo entonces no existe z' tal que $z \Longrightarrow^* z'$ (con $z \neq z'$)

Como z ε LNV, z es de forma (1) ó (2). Prueba del teorema por inducción sobre pc(z): si pc(z) = 1): si z es de forma $[z_1,\ldots,z_n]$: no se puede aplicar $\Longrightarrow$ sobre z (por ser infimo) ni $\Longrightarrow^*$ sobre cualquiera de sus argumentos es decir que todos son átomos ó variables, ó funciones con arg en forma normal sin listas y estan en forma normal, entonces por definición (3) z esta en formal normal.

Si z es de forma $[z_1,\ldots,z_n \mid z_{n+1}]$ por las mismas razones por definición (4) z esta en forma normal.

Se supone verdadero el teorema

para pc(z) = k

Mostremolo para pc(z) = k+1

Sí z es de la forma $z = [z_1,\ldots,z_n]$, $pc(z_i) \leq k \; \forall_i \leq n$ entonces cada z_i esta en forma normal y por definición (3) z también está en forma normal.

Sí z es de la forma $z = [z_1,\ldots,z_n \mid z_{n+1}]$ $pc(z_i) < k \; \forall_i \leq n+1$ entonces cada z_i esta en forma normal y z_{n+1} no es de ninguna de las formas

(8) $[\, y_1,\ldots,y_m \,]$

(9) $[\, y_1,\ldots,y_m \mid y_{m+1} \,]$

(10) $[\ \ ]$

por que sino se podría aplicar (5), (6) ó (7) sobre z y z no sería un ínfimo, entonces $z_{n+1} \notin L$

Luego z está en forma normal por definición (4).

Teorema $\forall \ell \in$ LNV existe $\ell' \in$ LFN tal que $\ell \Longrightarrow^* \ell$ y ℓ' es única.

Prueba Por las propiedades (1) y (2) se deduce que $\exists \ell' \in$ LFN tal que $\ell \Longrightarrow^* \ell'$

Mostremos que ℓ' es única.

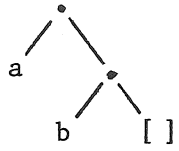
- por definición de $\Longrightarrow$ (partición en 3 casos mutualmente exclusivos) que se deduce que $\forall u \in$ LNV si existe v tal que $u \Longrightarrow v$ entonces v es única.
- por definición de $\Longrightarrow^i$ (partición en 3 casos mutualmente exclusivos) se deduce que $\forall u \in$ LNV si existen i y v tales que $u \Longrightarrow^i v$ entonces v es única.

La existencia de una forma normal única para cada lista nos permitira simplificar todas las manupulaciones formales sobre listas, trabajando con su forma normal.

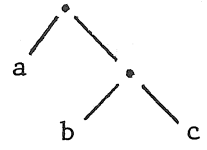
II. Las Expresiones Internas.

Las expresiones vistas anteriormente, tienen dos tipos de estructuras, las funciones y las listas; generalmente se acostumbra representar las listas mediante funciones de una forma muy sencilla:

La lista [a,b] se representa como la función .(a,.(b, [])) y la lista [a,b|c] como la función .(a,.(b,c)); estas, vistas en forma de árbol serían:



[a,b]



[a,b|c]

Con esto, el problema de representación, sólo toma en cuenta variables, átomos y funciones.

En el sistema que se propone, las funciones se representan con listas; por ejemplo, la función f(a,g(b)) se representan como [f,a,[g,b]] donde f y g son símbolos de funciones.

Sean los siguientes subconjuntos dos a dos disjuntos:

A de Atomos

V de Variables

F' de functores o símbolos de función

LNV de listas no vacías.

Una Fxpresión Interna es:

- un átomo
- una variable
- un functor
- una lista no vacía de expresiones internas.

Como las listas no vacías, no cambian en las EI, todo lo dicho sobre expresiones en forma normal, se aplica también con las internas.

A cada EEFN se le puede hacer corresponder una EIFN; algunos ejemplos son:

[f(X), g(Y)] se representa como [[f, X], [g, Y]]

[f(a), b|f(c)] se representa como [[f, a], b, f, c]

Es fácil ver que hay EI que no corresponden a ninguna EE, tal es el caso de la lista [f, g] donde f y g son functores. Sin embargo para aquellas EI a las que les corresponde alguna EE, se puede demostrar que sólo les corresponde una.

Función de transformación:

Sea ψ: EEFN → EIFN la función de transformación definida por:

(1) $\psi(a) = a$ si a es un átomo o una variable

(2) $\psi(f(e_1,\ldots,e_n)) = [f,\psi(e_1),\ldots,\psi(e_n)]$ donde $f(e_1,\ldots,e_n)$ es una función

(3) $\psi([e_1,\ldots,e_n]) = [\psi(e_1),\ldots,\psi(e_n)]$

(4) $\psi([e_1,\ldots,e_n|e]) = [\psi(e_1),\ldots,\psi(e_n)|\psi(e)]$ si e no es una función

(5) $\psi([e_1,\ldots,e_n|f(e_{n+1},\ldots,e_m)] = [\psi(e_1),\ldots\psi(e_n),f,\psi(e_{n+1}),\ldots,\psi(e_m)]$

Las transformaciones (1), (2) y (3) preservan las formas normales, la cuarta también si e no es una función, como es el caso; para las funciones al final de una lista se usa (5) que no es mas que (4) normalizada.

En (2) ψ hace corresponder a todas las funciones $f(e_1, \ldots, e_n)$, $n > 0$ el functor f, la diferencia entre por ejemplo la función f con un argumento y la función f con dos es el número de elementos que quedan en la lista, luego del functor.

Para ver que ψ es inyectiva, sólo hace falta ver que es inyectiva para el caso de las funciones, puesto que para las otras expresiones se comporta como la identidad:

Sean $f(e_1,\ldots,e_n)$ y $f'(e'_1,\ldots,e'_m)$ dos funciones cualesquiera, tales que $\psi(f(e_1,\ldots,e_n)) = \psi(f'(e'_1,\ldots,e'_m))$ por la definición de ψ, es evidente que $n = m$; por inducción sobre las funciones:

1. Si $e_1,\ldots,e_n$ y $e'_1,\ldots,e'_n$ no contienen funciones:

   $[f, \psi(e_1),\ldots,\psi(e_n)] = [f', \psi(e'_1),\ldots, \psi(e'_n)] \Rightarrow f = f'$ y

   $e_1 = e'_1, \ldots, e_n = e'_n$ lo cual no significa más que $f(e_1,\ldots, e_n) =$

   $= f'(e_1, \ldots, e_n)$

K. Si $\psi(e_1) = \psi(e'_1), \ldots, \psi(e_n) = \psi(e'_n) \Rightarrow e_1 = e'_1,\ldots, e_n = e'_n$ por tanto, por el mismo argumento anterior

$$f(e_1, \ldots, e_n) = f'(e'_1, \ldots, e'_n)$$

III. Unificación de Expresiones en forma normal.

Definición: Una sustitución π en un par $[v_1,\ldots,v_n]\backslash[x_1,\ldots,x_n]$ formado por una lista de variables distintas entre sí y una lista de expresiones (de la misma longitud que la de las expresiones), que representan la lista de expresiones con las que cada variable se sustituye. Estas expresiones cumplen con una restricción:

(R): La expresión x_i correspondiente a la variable v_i en una sustitución π puede ser de dos formas:

(a) La misma v_i, indicando que no hay sustitución para ella.

(b) Una expresión finita cuyas variables pertenecen todas a la lista de variables de π y tienen todas una sustitución de la forma (a).

Definición: La transformación ψ se extiende para las sustituciones (externas), de la siguiente forma:

$$\psi([v_1,\ldots,v_n]\backslash[x_1,\ldots,x_n]) = [v_1,\ldots,v_n]\backslash[\psi(x_1,\ldots,\psi(x_n)]$$

que es una sustitución Interna.

Definición: Una sustitución π es aplicable a una expresión e, si todas las variables de e, están todas en la lista de variables de π.

Definición: La aplicación $e.\pi$ de una sustitución π aplicable a una expresión e se define de la siguiente forma:

(1) $a.\pi = a$ si a es un átomo

(2) $v_i.\pi = x_i$ si v_i es una variable, x_i es una expresión y $\pi = [v_1,\ldots,v_i,\ldots,v_n]\backslash[x_1,\ldots,x_i,\ldots,x_n]$

(3) $[e_1,\ldots,e_n] \cdot \pi = [e_1.\pi,\ldots,\ldots,e_n.\pi]$

(4) $[e_1,\ldots,e_n|e] \cdot \pi = [e_1.\pi,\ldots,e_n.\pi|e.\pi]$; en el caso en que $e.\pi$ sea una lista, hay que normalizar. Y una de las reglas siguientes, dependiendo del tipo de expresión es:

($\mathfrak{S}$.E) $f(e_1,\ldots,e_n).\pi = f(e_1.\pi,\ldots,e_n.\pi)$ donde f es el nombre de una función cualquiera de aridad n.

($\mathfrak{S}$.I) $f.\pi = f$ donde d es un fuctor.

Definición: Una sustitución π es un unificador para dos expresiones e_1 y e_2 si

(a) e_1 y e_2 no tienen variables distintas con el mismo nombre.

(b) π es aplicable tanto a e_1 como a e_2 y

(c) $e_1.\pi = e_2.\pi$

Definición Dos expresiones e_1 y e_2 son unificables si existe un unificador π para e_1 y e_2.

Ejemplo: Sean $e_1 = f(A, g(B, A))$ y $e_2 = f(h(C), D)$ un unificador para e_1 y e_2 puede ser, por ejemplo:

$$\pi = [A, B, C, D]\backslash[h(c), B, C, g(B, h(c))]$$

de manera que:

$$e_1.\pi = e_2.\pi = f(h(c), g(B, h(c)))$$

Todas estas definiciones son válidas, tanto para las EI como para las EE y más aún, se puede demostrar que la transformación ψ preserva la unificación:

Teorema: Sean e_1 y e_2 dos EE y π un unificador para ellas, entonces:

$$e_1.\pi = e_2.\pi \Rightarrow \psi(e_1).\psi(\pi) = \psi(e_2).\psi(\pi)$$

Para la demostración, se usará la proposición:

$$\psi(e.\pi) = \psi(e) \ . \ \psi(\pi)$$

Demostración del Teorema:

$$e_1.\pi = e_2.\pi \Rightarrow \psi(e.\pi) = \psi(e_2.\pi)$$
$$\Rightarrow \psi(e_1).\psi(\pi) = \psi(e_2).\psi(\pi)$$

Demostración de la proposición:

Haciendo inducción sobre listas y funciones:

1. Si a es un átomo

$\psi(a.\pi) = \psi(a) = \psi(a) \ . \ \psi(\pi)$

2. Si e es una variable V_i

$\psi(V_i . [V_1,\ldots,V_i,\ldots,V_n]\backslash[X_1,\ldots,X_i,\ldots,X_n]) = \psi(X_i)$ y

$\psi(V_i).\psi([V_1,\ldots,V_i,\ldots,V_n]\backslash[X_1,\ldots,X_i,\ldots,X_n]) =$

$V_i[V_1,\ldots,V_i,\ldots,V_n]\backslash[\psi(X_1),\ldots,\psi(X_i),\ldots,\psi(X_n)] = \psi(X_i)$

3. Si e es una lista (a)$[e_1,\ldots,e_n]$ ó (b)$[e_1,\ldots,e_{n-1}|e_n]$ tal que $\psi(e_i.\pi) = \psi(e_i).\psi(\pi)$, $i = 1,\ldots,n$

(a) $\psi([e_1,\ldots,e_n].\pi) = \psi[e_1.\pi,\ldots,e_n.\pi] =$

$[\psi(e_1.\pi),\ldots,\psi(e_n.\pi)] =$

$[\psi(e_1).\psi(\pi),\ldots,\psi(e_n).\psi(\pi)] =$

$[\psi(e_1),\ldots,\psi(e_n)].\psi(\pi) = \psi([e_1,\ldots,e_n]).\psi(\pi)$

(b) Se demuestra análogamente

4. Si e es una función cualquiera de aridad n $f(e_1,\ldots,e_n)$ tal que $\psi(e_i.\pi) = \psi(e_i).\psi(\pi)$ $i = 1,\ldots,n$

$\psi(f(e_1,\ldots,e_n).\pi) = \psi(f(e_1.\pi,\ldots,e_n.\pi)) =$

$[f,\psi(e_1.\pi),\ldots,\psi(e_n.\pi)] =$

$[f.\psi(e_1), \psi(\pi), \ldots\ldots,\psi(e_n).\psi(\pi)] =$

$[f.\psi(\pi),\psi(e_1).\psi(\pi),\ldots,\psi(e_n).\psi(\pi)] =$

$[f,\psi(e_1),\ldots,\psi(e_n)].\psi(\pi) =$

$\psi(f(e_1,\ldots,e_n)).\psi(\pi).$

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

CAPITULO B

PROGRAMACION, LENGUAJES Y COMPILADORES

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

ASIGNACION CONCURRENTE DE VECTORES Y MATRICES

Juan Carlos ANSELMI
Centro de Informaciones
y Estudios del Uruguay (CIESU)

1. INTRODUCCION

En ALGOL-60, y en muchos otros lenguajes de programación, para intercambiar los valores del J-ésimo y del K-ésimo elemento de un vector "S", es necesario utilizar una variable adicional (por ejemplo "Y") que sea del mismo tipo que los elementos de dicho vector.

```
Y := S(J)  ;  S(J) := S(K)  ;  S(K) := Y  ;
```

Sería mucho más claro y sencillo si esta permutación de valores se pudiera efectuar con una única sentencia, en donde a la izquierda del símbolo ":=" se especificara de alguna manera la pareja ordenada S(J) , S(K) , y a la derecha se especificara la pareja ordenada S(K) , S(J).

Este tipo de asignación múltiple, que permite definir varios elementos de un vector o de una matriz, presenta sin embargo algunos problemas, especialmente cuando se utilizan matrices multi-indizadas, o cuando en el miembro de la derecha se usa también la matriz indicada en el miembro de la izquierda.

E. W. Dijkstra opina que la causa de estos problemas no está en la asignación concurrente sino en la noción de variable indizada.

> *"However, I have now come to the conclusion that it is not the concurrent assignment, but the notion of the subscripted variable that is to be blamed"* (cf. DIJKSTRA-76, capítulo 11, página 95).

El objetivo del presente trabajo es precisamente el de revisar el concepto de matriz, definiendo con precisión y claridad las manipulaciones que serán permitidas y los valores de los índices que podrán ser usados, a efectos de posibilitar la aplicación de la asignación concurrente.

El uso de algún tipo de asignación concurrente en los lenguajes de programación o en los lenguajes de especificación, permite generalmente expresar los algoritmos de cálculo más brevemente y con mayor claridad.

Supongamos, por ejemplo, que se tiene un vector "P" cuyo índice varía entre 1 y 60, y que se desea reubicar todos los valores de elementos correspondientes a valores impares del índice, colocándolos en forma correlativa a partir del elemento P(11) y hasta el elemento P(40).

Esta transformación puede ser efectuada con el siguiente algoritmo:

```
L := 20  ;  M := 19  ;

MIENTRAS  L > 10  REPETIR : P(L) := P(M) ;
                            L := L - 1 ;
                            M := M - 2 ;
L := 22  ;  M := 23  ;
MIENTRAS  L < 41  REPETIR : P(L) := P(M) ;
                            L := L + 1 ;
                            M := M + 2 .
```

Aplicando el sistema propuesto en el presente trabajo, se puede indicar esta misma transformación del vector "P" con la siguiente asignación concurrente:

```
P(11:40) := P(1:59:2) .
```

Esta última sentencia es evidentemente mucho más corta que el algoritmo antes detallado.

La economía de líneas de texto del programa cuando se utiliza la asignación concurrente, puede llegar a ser bastante importante, especialmente cuando se utilizan matrices de dos o más índices.

La asignación concurrente abre también nuevas perspectivas (y muy interesantes por cierto) en las computadoras que permiten desarrollar en paralelo varias tareas.

En efecto, en este caso, el programador podría continuar programando en la forma secuencial habitual, y la máquina, en forma automática, podría introducir el paralelismo a nivel interno de cada sentencia de asignación concurrente, ya que una parte de la copia de valores podría ser efectuada por un procesador, y la parte restante podría ser realizada paralelamente por otro procesador.

2. VALORES POSIBLES DE UN INDICE

Los valores permitidos de un índice serán indicados en forma abreviada por tres parámetros enteros, los cuales serán especificados separándolos con el carácter ":". El conjunto índice A:B:V estará compuesto por los valores enteros A , A+V , A+V+V , A+V+V+V , ... , B-V-V , B-V , y B.

Se admitirá que "A" pueda ser menor, igual o mayor que el valor "B". Además, si "A" es igual a "B", el valor "V" podrá ser cualquier valor entero, pero si "A" es distinto de "B", entonces el valor "V" deberá ser siempre entero, no nulo, del mismo signo que (B-A), y tal que el valor absoluto de (B-A) sea un múltiplo entero del valor absoluto de "V".

Dado el conjunto de valores posibles A:B:V de un índice, diremos que "A" es el primer valor índice, que "B" es el último valor índice, y que "V" es la variación del índice.

El conjunto índice A:B:V, con "B" estrictamente mayor que "A" y "V" igual a la unidad, será indicado abreviadamente por A:B. Cuando "B" es estrictamente menor que "A" y "V" es igual a -1, entonces el conjunto índice A:B:V también será indicado abreviadamente por A:B.

Además, A:A:V será indicado abreviadamente por A:A, cualquiera sean los valores enteros "A" , "V".

Evidentemente, si "A" es igual a "B", el conjunto índice A:B:V estará compuesto por un único valor índice; y si "A" es distinto de "B", el conjunto índice A:B:V estará formado por n = (B-A+V)//V valores índice (la división entera ha sido indicada con el símbolo "//") . La función que expresa el número de valores diferentes de un conjunto índice será indicada en lo sucesivo con el nombre "dim" .

Si $A \neq B$, entonces : (A:B:V).dim = (B-A+V)//V

Si $A = B$, entonces : (A:B:V).dim = 1

Conviene aclarar que un índice está caracterizado por los valores que puede adoptar, y por el orden creciente o decreciente en el cual dichos valores son considerados. En este sentido, el conjunto índice A:B:V es generalmente diferente del conjunto índice B:A:-V , pues si bien ambos definen los mismos valores índice, difieren en el orden en el cual dichos valores índice son considerados en el caso que"A" sea distinto de "B".

Ya hemos dicho que no todas las ternas de parámetros enteros caracterizan sin ambigüedad un conjunto de valores índice, pues se requieren ciertas condiciones. Dichas condiciones pueden ser verificadas con la función "test", que al aplicarla a la terna A:B:V proporciona el valor lógico ".verdadero." cuando "A" es igual a "B" , o cuando siendo "A" distinto de "B" , se cumple que "V" es no nulo, del mismo signo que (B-A), y tal que el valor absoluto de (B-A) es un múltiplo entero del valor absoluto de "V" ; en caso contrario, la función "test" proporciona el valor lógico ".falso.".

Al aplicar la misma función "test" a una terna A:B:V y a un valor entero "C", se puede determinar si dicho valor entero es uno de los valores índice del conjunto A:B:V. En este caso, (A:B:V).test(C) será igual a ".verdadero." si y sólo si se cumplen las siguientes tres condiciones:

a) se verifica que (A:B:V).test = .verdadero. ;

b) el valor absoluto de (C-A) es igual a cero o es un múltiplo entero del valor absoluto de "V" ;

c) el valor (C-A)*(C-B) es siempre negativo o nulo.

Es posible también determinar si todos los índices del conjunto índice C:D:W son también índices del conjunto índice A:B:V , aplicando la función "test" a estas dos ternas ; esta aplicación de la función "test" está evidentemente orientada a la utilización de subconjuntos índice.

Concretamente, (A:B:V).test(C:D:W) será igual al valor lógico ".verdadero." si y sólo si se cumplen las tres condiciones siguientes:

a) (A:B:V).test = .verdadero. ;

b) (C:D:W).test = .verdadero. ;

c) Si (C:D:W).test(E) = .verdadero. , entonces también se cumple (A:B:V).test(E) = .verdadero. .

Detalladamente, las condiciones b) y c) anteriores son equivalentes a las siguientes:

d1) El valor absoluto de (C-A) es igual a cero o es un múltiplo entero del valor absoluto de "V" ;

d2) El valor (C-A)*(C-B) nunca es estrictamente mayor que cero ;

d3) Se verifica una de las siguientes dos situaciones :

1er. caso

1) Se cumple que "C" es igual a "D" ;

2do. caso

2a) Se cumple que "C" y "D" no son iguales ;

2b) El valor "W" es no nulo y del mismo signo que el valor (D-C) ;

2c) El valor absoluto de (D-C) es múltiplo entero del valor absoluto de "W" ;

2d) El valor absoluto de "W" es múltiplo entero del valor absoluto de "V" ;

2e) El valor (D-A)*(D-B) nunca es estrictamente mayor que cero .

A partir de estas definiciones, se posible deducir varias propiedades de la función "test" , entre ellas las siguientes:

1) Si (A:B:V).test , entonces : (B:A:-V).test ;
2) Si (A:B:V).test(C) , entonces : (A:B:V).test ;
3) Si (A:B:V).test(C) , entonces : (A:C:V).test ;
4) Si (A:B:V).test(C) , entonces : (C:B:V).test ;
5) Si (A:B:V).test , entonces : (A:B:V).test(A) ;
6) Si (A:B:V).test , entonces : (A:B:V).test(B) ;
7) Si (A:B:V).test , entonces : (A:B:V).test(A:B:V) ;
8) Si (A:B:V).test , entonces : (A:B).test(A:B:V) ;
9) Si (A:B:V).test , entonces : (A:B:V).test(B:A:-V) ;
10) Si (A:B:V).test(C:D:W) , entonces : (A:B:V).test(C) ;
11) Si (A:B:V).test(C:D:W) , entonces : (A:B:V).test(D) ;
12) Si (A:B:V).test(C:D:W) , entonces : (A:B:V).test ;
13) Si (A:B:V).test(C:D:W) , entonces : (C:D:W).test ;
14) Si (A:B:V).test , entonces : (A:B:V).test(A:A) ;
15) Si (A:B:V).test , entonces : (A:B:V).test(B:B) ;

16) Si (A:B:V).test(C) , entonces : (A:B:V).test(A:C:V) ;

17) Si (A:B:V).test(C) , entonces : (A:B:V).test(B:C:-V) ;

18) Si (A:B:V).test , entonces : (A:B:V).test(A:B:B-A) ;

19) Si (A:B:V).test , entonces : (A:B:V).test(B:A:A-B) ;

20) Si (A:B:V).test(C:D:W) , entonces : (A:B:V).test(D:C:-W) .

3. EL CONCEPTO DE MATRIZ Y LAS FUNCIONES QUE LE SON APLICABLES

Una matriz estará caracterizada por uno o varios conjuntos índice del tipo de los definidos en la sección 2. , por el conjunto de los valores posibles de sus elementos, y por una función de acceso con dominio en el producto cartesiano de los conjuntos índice asociados a la matriz, y con alcance en el conjunto de valores posibles de los elementos de la matriz ; esta función de acceso puede permanecer invariable durante todo el proceso del programa o de la tarea, o puede ser total o parcialmente modificada por una sentencia de asignación o por una operación de entrada, en uno o en varios puntos del desarrollo del algoritmo.

A efectos de simplificar la presentación de este trabajo, en lo que sigue supondremos que todos los vectores y las matrices que se utilizarán serán del tipo "T", mientras que las variables elementales podrán ser del tipo "T" o de tipo entero, y en este último caso serán utilizadas como variables índice o como valores particulares de un índice.

Este tipo "T" podrá ser, por ejemplo, el tipo de datos "real", o el tipo "caracter", o el tipo "lógico", o cualquier otro tipo de datos previsto en el lenguaje de programación a utilizar.

Una matriz multi-indizada será creada por una función especial de creación, que en lo sucesivo llamaremos "crea".

Por ejemplo, U.crea(1:8) permite crear un vector llamado "U" cuyo único índice puede adoptar cualquier valor entero no menor que el valor unidad y no mayor que "8".

Por su parte, R.crea(1:7:2,9:-9) permite crear una matriz bi-indizada "R" cuyo primer índice puede tomar los valores 1, 3, 5, 7 , y cuyo segundo índice puede tomar los valores enteroscomprendidos entre -9 y 9.

En general, S.crea(A:B:V) permite crear un vector "S" cuyo único índice puede adoptar los valores del conjunto índice A:B:V , Z.crea(A:B:V,AA:BB:VV) permite crear una matriz "Z" cuyos dos índices pueden tomar respectivamente alguno de los valores definidos por los conjuntos índice A:B:V , AA:BB:VV , y análogamente para las matrices tri-indizadas y las otras matrices multi-indizadas.

Puesto que parece razonable que las variables elementales tales como "X" o "Y" también puedan ser análogamente creadas con X.crea e Y.crea, es conveniente considerar a dicho tipo - de variables como un caso particular de matriz multi-indiza-- da, las cuales no tendrían asociado ningún índice ; en este - caso, diremos que se trata de matrices elemento. Nótese que - la función de acceso asociada a una matriz de este tipo, permite obtener el valor correspondiente al único punto anónimo del dominio.

Obsérvese que no hemos especificado los valores de los elementos de las matrices creadas de esta forma, pues se supone que dichos valores serán posteriormente definidos por una o varias sentencias de asignación o por otro procedimiento conveniente ; inicialmente, todos los elementos de una matriz podrán ser definidos en forma automática con un valor estandar (por ejemplo, cero, .falso. , etc. , o con un valor especial que señale la indefinición) .

El número de índices de una matriz multi-indizada puede hacerse accesible con una función llamada "dom" ; para las ma-- trices de los ejemplos anteriores, se cumplirá :

```
U.dom = 1  ;  R.dom = 2  ;  S.dom = 1  ; Z.dom = 2  ;
X.dom = 0  ;  Y.dom = 0  .
```

Un índice particular de una matriz multi-indizada podrá -- ser representado por el número de orden correspondiente antecedido por el signo "?" ; los diferentes índices de una ma- - triz "Q" serán así identificados por ?1 , ?2 , ?3 , etc., hasta ?(Q.dom) .

El conjunto índice asociado a cierto índice de una matriz puede ser accedido con una función de acceso llamada "ci" ; para las matrices "U" , "R" , "Z" anteriormente creadas, se tendrá:

```
U.ci(?1) = 1:8 ; R.ci(?1) = 1:7:2 ; S.ci(?1) = A:B:V ;
Z.ci(?1) = A:B:V ; Z.ci(?2) = AA:BB:VV .
```

Por su parte, puede accederse también al primer índice, al último índice, y a la variación de un índice de una matriz - multi-indizada, con las funciones de acceso "pi" , "ui" , "vi" ; estas funciones pueden ser directamente aplicadas a una matriz, o aplicadas al resultado proporcionado por la función "ci" . Por ejemplo:

```
U.ci(?1).pi = U.pi(?1) = 1  ;U.ci(?1).vi = U.vi(?1) = 1 ;
R.ci(?2).ui = R.ui(?2) = -9 ;R.ci(?2).vi = R.vi(?2) = -1 ;
Z.ci(?1).pi = Z.pi(?1) = A  ;Z.ci(?1).ui = Z.ui(?1) = B ;
U.ci(?1).ui = U.ui(?1) = 8  ;R.ci(?1).vi = R.vi(?1) = 2 ;
```

R.ci(?2).pi = R.pi(?2) = 9 ; S.ci(?1).vi = S.vi(?1) = V .

Análogamente se utilizará la función "dim", que expresa el número de valores diferentes de un índice.

U.ci(?1).dim = U.dim(?1) = 8 ; R.ci(?2).dim = R.dim(?2) = 19

Es posible considerar todos o una parte de los elementos de una matriz, estructurándolos de tal forma de obtener una nueva matriz ; diremos que esta nueva matriz es una submatriz de la anterior.

Una submatriz de la matriz "Z" quedará perfectamente definida si, para cada índice de "Z" , se especifica el único valor índice que será considerado, o si se especifica el subconjunto índice que será tomado en cuenta.

El número de índices de una submatriz de "Z" será pues -- igual al número de subconjuntos índice utilizados ; evidentemente, dicho valor nunca podrá exceder al número de índices - de la propia matriz "Z".

Una submatriz de la matriz "Z" será especificada utilizando una función especial que llamaremos "sub" . Dicha función se aplicará a la matriz original y a la lista de valores índice y de conjuntos índice que caracterizan a la submatriz.

Se indican a continuación varios ejemplos de submatrices - de las matrices "U", "R" antes definidas.

U.sub(4) ; U.sub(1:8) ; U.sub(1:4) ; U.sub(1:7:2) ;

U.sub(8:1) ; U.sub(5:5) ; U.sub(8:2:-2) ; U.sub(8) ;

R.sub(5:5,0:9) ; R.sub(3,9:1:-2) ; R.sub(1:7:6,0:0) ;

R.sub(5:1:-4,-9:9:2) ; R.sub(1,0) ; R.sub(7:7,-9:-9) .

Obsérvese que la submatriz U.sub(4:4) es diferente de la submatriz U.sub(4), ya que la primera submatriz tiene un único índice, mientras que la segunda no tiene ninguno.

Obsérvese también que la propia matriz original es siempre submatriz de sí misma.

Evidentemente, a efectos de definir correctamente una submatriz de una matriz "Z", se deberá especificar para cada índice de "Z", o bien un valor índice factible, o bien un subconjunto índice del conjunto índice originalmente asociado a ese índice de "Z" ; por lo tanto, si aplicamos la función -- "test" a este último conjunto índice, y al valor índice o al subconjunto índice correspondiente y asociado a la submatriz que se desea definir, se debe obtener siempre el resultado -- ".verdadero." si la submatriz es viable. Parece entonces natural introducir un mecanismo que permita efectuar este control

en forma concurrente para todos y cada uno de los índices de una matriz.

Dicho mecanismo será implementado permitiendo la aplicación de la función "test" a una matriz y a una lista de valores índice y de conjuntos índice. Deberá suponerse en este caso que la función "test" se aplica a cada conjunto índice de la matriz, y al valor índice o al conjunto índice que le corresponda en la lista. El resultado global de la función "test" será igual a ".verdadero." si y sólo si cada una de estas evaluaciones parciales proporciona el resultado ".verdadero." .

Evidentemente, si "S" es un vector, entonces se cumplen las siguientes relaciones:

```
S.test(C) = S.ci(?1).test(C)
S.test(C:D:W) = S.ci(?1).test(C:D:W)
```

Los siguientes ejemplos de aplicación de la función "test" a las matrices "U", "R" antes definidas dan todos como resultado el valor lógico ".verdadero." .

```
U.test(7:7) ; U.test(8:1:-7) ; U.test(2:5) ; U.test(2) ;
U.test(6:4) ; R.test(7:1:-2,9:-9:-2) ; R.test(5,-7:7) ;
R.test(7:3:-4,-7) ; R.test(5:5,4:4) ; R.test(5,-7:-3) .
```

Por el contrario, las siguientes evaluaciones dan todas como resultado el valor lógico ".falso." .

```
R.test(7:3:2,5:-8) ; R.test(5,9:0:-2) ; R.test(0,0) ;
R.test(9:-9,1:7:2) ; R.test(1:7:3,-9:9:3) ; U.test(9) ;
U.test(1:8:2) ; R.test(7:1,7:1) ; R.test(7:3:1.7:1) .
```

Obsérvese que la función "sub" permite especificar una subestructura de datos totalmente contenida en una matriz multiindizada. Esto puede ser necesario a efectos de re-definir dicha subestructura con una sentencia de entrada de información o con una sentencia de asignación, o a efectos de especificar un argumento de salida o de entrada/salida en una invocación a un subprograma.

En otros casos, sólo interesará acceder a los valores de los elementos de una submatriz, y no a la subestructura misma ; a efectos de permitir este acceso, será necesario introducir una nueva función que llamaremos "val" .

La función "val" también se aplicará a una matriz y a una lista de valores índices y de conjuntos índice, en forma similar a lo que se efectuaba con la función "sub".

Nótese que las funciones "sub" y "val" permiten expresar una sentencia de asignación de una manera mucho más explícita.

Evidentemente, es mucho más significativa para el programador una sentencia como U.sub(2) := U.val(8) , que la usual y más desprolija sentencia de asignación U(2) := U(8) .

Claro que puede argumentarse que esta última forma es una abreviación de la primera, lo cual es cierto. Esto no desmerece la utilidad de las funciones "sub" y "val" , no sólo desde el punto de vista conceptual, sino por las grandes ventajas que aportan al utilizarlas para indicar los argumentos en una invocación a un subprograma.

En efecto, con el uso de dichas funciones, resulta totalmente innecesario plantear a nivel práctico los clásicos dos tipos de transmisión de argumentos (por valor y por nombre) que son causa frecuente de errores de programación. Además, ello permite que se puedan efectuar ciertos controles suplementarios durante la compilación del programa, ya que será incorrecto utilizar la función "val" cuando el argumento es de salida o de entrada/salida, y será innecesario usar la función "sub" cuando el argumento es de entrada. Evidentemente, estos controles podrán ser efectuados automáticamente por el compilador, si el lenguaje de programación utilizado permite declarar los distintos tipos de argumentos formales empleados.

Diremos que dos matrices son homólogas cuando tienen igual número de índices, y cuando sus índices correspondientes tienen igual dimensión.

Por lo tanto, dos matrices "Z", "ZZ" serán homólogas si y sólo si se cumplen las siguientes condiciones:

(a) Z.dom = ZZ.dom

(b) Z.dim(?k) = ZZ.dim(?k) para k en (1..(Z.dom))

El conjunto de valores enteros comprendidos entre 1 y el valor Z.dom ha sido indicado abreviamente por (1..(Z.dom)) ; debe entenderse que dicho conjunto es vacío si Z.dom = 0 .

Se indican seguidamente algunos ejemplos de matrices que son homólogas.

(1) U.sub(7:4) y R.sub(1:7:2,4)

(2) Z.sub(A,BB) , U.sub(5) , X.sub e Y.sub

(3) Z.sub(B,BB:BB) y U.sub(3:3)

(4) S.sub(A:B:V) y Z.sub(B:A:-V,AA)

(5) R.sub(7:3:-2,0:9) y R.sub(1:5:2,-9:9:2)

Evidentemente, la homología entre matrices es una relación de equivalencia.

Diremos que dos matrices "Z" , "ZZ" son homólogas en el sentido amplio, cuando se cumple la relación Z.dim(?k) igual a ZZ.dim(?k) para todo valor entero positivo de k que sea menor o igual que el máximo de los valores Z.dom , ZZ.dom . Debe entenderse que si k es estrictamente mayor que Z.dom , entonces debe sustituirse en la relación anterior Z.dim(?k) por el valor unidad ; análogamente, si k es estrictamente mayor que ZZ.dom , entonces debe operarse la sustitución de ZZ.dim (?k) por el valor unidad.

La homología en el sentido amplio también es una relación de equivalencia definida en el conjunto de las matrices.

Evidentemente, la matriz elemento S.sub(A) es homóloga en el sentido amplio al vector U.sub(3:3) y al vector R.sub(5,9:9) .

4. ASIGNACION CONCURRENTE DE UNA MATRIZ CON LOS VALORES DE OTRA MATRIZ.

La definición concurrente de los elementos de una matriz con los valores de los elementos de otra matriz sólo será posible cuando ambas matrices son homólogas en el sentido amplio.

La asignación se hará elemento a elemento ; cada elemento de la matriz indicada en el miembro izquierdo de la sentencia de asignación será definido con el valor del elemento correspondiente del otro miembro.

Por ejemplo, U.sub(1:8:7) := U.val(8:1:-7) permite permutar los valores de los elementos U(1) y U(8).

Por su parte, U.sub(1:8) := U.val(8:1) es equivalente al algoritmo detallado de asignación que se indica a continuación.

```
LA := 4 ; LB := 5 ;

MIENTRAS  LA > 0  REPETIR:

          Y := U(LA) ;
          U(LA) := U(LB) ;
          U(LB) := Y ;
          LA := LA - 1 ;
          LB := LB + 1 .
```

En el caso U.sub(2:3) := R.val(3,0:-3:-3) , el elemento U(2) es definido con el valor de R(3,0) , y el elemento U(3) con el valor de R(3,-3).

5. COMPOSICION DE VALORES

Las constantes y los valores de los elementos de las matrices pueden componerse para formar estructuras de datos mayores, utilizando los siguientes operadores de composición:

a) el operador "," para la composición al nivel del primer índice ;

b) el operador ";" para la composición al nivel del segundo - índice ;

c) el operador ";;" para la composición al nivel del tercer índice ;

d) en general, el operador ";?k;" para la composición al nivel del k-ésimo índice (este operador incluye como casos - particulares a los tres anteriores) .

Para poder componer los valores de una matriz "Z" con los valores de una matriz "H" al nivel del k-ésimo índice, se debe verificar la siguiente condición :

a) Se debe cumplir que Z.dim(?j) = H.dim(?j) para todos los - valores de j diferentes de k y menores o iguales que el máximo de los valores Z.dom , H.dom . Debe entenderse que si j es estrictamente mayor que Z.dom , entonces en la relación anterior debe sustituirse Z.dim(?j) por el valor unidad ; análogamente, si j es estrictamente mayor que H.dom, entonces debe operarse la sustitución de H.dim(?j) por el valor unidad.

Esta composición produce una estructura de datos de tipo - matricial, que llamaremos Z;?k;H , y que tiene las siguientes características :

a) El valor (Z;?k;H).dom es igual al máximo de los tres valores k , Z.dom , H.dom .

b) Se cumple (Z;?k;H).dim(?j) = Z.dim(?j) = H.dim(?j) para todos los valores de j diferentes de k y menores o iguales - que el valor (Z;?k;H).dom ; en la relación anterior, debe sustituirse Z.dim(?j) y/o H.dim(?j) por el valor unidad, toda vez que exista indefinición de dichos valores.

c) Se verifica que (Z;?k;H).dim(?k) = Z.dim(?k) + H.dim(?k) ; debe suponerse que si k es estrictamente mayor que el valor Z.dom, entonces en la relación anterior debe sustituirse Z.dim(?k) por el valor unidad (y análogamente para - - H.dim(?k)).

En todo lo anterior, se ha supuesto que tanto j como k son siempre valores enteros positivos.

La composición (Z;?k;H) produce pues una estructura matricial de valores de las características antes indicadas, en -- donde para las primeras ocurrencias del k-ésimo índice se tienen los valores de los elementos de la matriz "Z" , y para - las últimas ocurrencias del k-ésimo índice se tienen los valores de los elementos de la matriz "H" .

La composición puede también operarse entre valores de submatrices, entre constantes, o entre constantes y valores de - submatrices ; en estos últimos dos casos, las constantes serán asimiladas a matrices elemento.

Los distintos operadores de composición pueden aplicarse - en forma reiterada ; en este caso, la utilización de paréntesis permitirá evitar ambigüedades.

Por ejemplo, (0 , 1 , 28 , -2) forma un vector de cuatro elementos.

Por su parte, el siguiente ejemplo forma una matriz de dos índices, el primero de los cuales tiene dimensión dos y el segundo dimensión tres.

((1 , 2) ; (11 , 12) ; (21 , 22))

La misma estructura, con los mismos valores,puede ser también expresada de la forma que se indica a continuación.

((1 ; 11 ; 21) , (2 ; 12 ; 22))

La composición (0.0 ; 1.0) forma una matriz tri-indizada cuyo primer índice tiene dimensión 1 , su segundo índice tiene también dimensión 1 , y su último índice tiene dimensión - 2 .

6. LA ASIGNACION CONCURRENTE EN EL CASO GENERAL

El miembro izquierdo será siempre una matriz o una submatriz. El miembro derecho será una composición de valores.

Las dimensiones de los índices correspondientes de ambos - miembros deben ser iguales entre sí, y las dimensiones de los índices que no tengan correspondiente en el otro miembro deben ser iguales a la unidad.

La asignación se hará elemento a elemento. Cada elemento - de la matriz indicada en el miembro izquierdo será definido - con el valor correspondiente indicado en el otro miembro.

A efectos de que el algoritmo que permita efectuar la asignación concurrente sea razonablemente simple, no se permitirá

utilizar en el miembro derecho más de una submatriz correspondiente a la matriz indicada en el miembro izquierdo.

Por ejemplo, la sentencia

```
P.sub(1:3:2,0:2:2) := (R.val(3:1:-2,2) ; ( 5.0 , 8.1 ) )
```

asigna el valor de R(3,2) al elemento R(1,0) , el valor de R(1,2) al elemento R(3,0) , el valor 5.0 al elemento R(1,2) , y el valor 8.1 al elemento R(3,2) .

7. CONCLUSIONES

Opinamos que los conceptos vertidos son importantes en varios aspectos.

Para el programador de aplicaciones, la asignación concurrente le permitirá plantear algoritmos más cortos y claros (aunque más no sea en un lenguaje de especificación y como etapa previa a la codificación del programa en un lenguaje de programación), y las funciones "sub" y "val" le permitirán expresar los argumentos de los subprogramas de una manera más racional.

Para el investigador que se preocupa de la definición de nuevos lenguajes de programación, el nuevo concepto de matriz introducido en este trabajo lo llamará a la reflexión sobre las ventajas que dicho concepto aporta, permitiéndole posiblemente apreciar más claramente los problemas que puede causar la utilización de este tipo de estructura de datos.

8. BIBLIOGRAFIA

(DIJKSTRA-76) - A discipline of programming - E.W. Dijkstra - Prentice-Hall, Englewood Cliffs (New Jersey), 1976.

(ANSELMI-78a) - Sous-structures d'un tableau - J.C. Anselmi - Electricidad de Francia, Clamart (Francia), Publicación EDF HI 2695/02, febrero de 1978.

(ANSELMI-78b) - An algorithm descriptive language, the "LEAL" Language : treatment of data structures - J.C. Anselmi - Electricidad de Francia, Clamart (Francia), agosto de 1978.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

UM METODO DE PROGRAMAÇÃO BASEADO NO MODELO DATA-FLOW

Gentil J. de Lucena Filho
Depto. de Sistemas e Computaçao
Universidade Federal da Paraíba

Maarten H. van Emden
Dept. of Computer Science
University of Waterloo

1. INTRODUÇÃO

Em programação, como em outras atividades, é útil distinguir-se entre fins e meios. Quando se escreve um programa, o fim é uma certa relação entre entrada e saída a ser computada pelo programa. Os meios são certas ações primitivas que, devidamente sequenciadas pelo programa, estabelecerão o fim desejado. No programa, o controle deste sequenciamento tanto pode ser especificado explicitamente (como, por exemplo, através de GO TO's) quanto implícitamente (como, por exemplo, através de WHILE's e IF's). Neste artigo, a palavra "controle" será usada para referir-se ao controle de sequenciamento de ações.

Um dos objetivos em metodologia de programação é possibilitar a produção de programas de cada vez mais alta qualidade, verificáveis, com uma quantidade de esforço, previsível, cada vez menor. Controle, há muito, tem sido identificado como uma fonte de dificuldade em programação. E.W. Dijkstra, por exemplo, é conhecido por ter chamado atenção para as desvantagens inerentes à especificação explícita de controle comparada à sua especificação implícita em construtos de programação gol-orientados (isto é, fim-orientados). Uma forma de melhorar métodos de programação é facilitar a especificação do fim (isto é, a relação entrada/saída) e requerer menos atenção aos meios (isto é, controle) através dos quais aquele é conseguido.

Ainda recentemente, Kowalski [6] caracterizou de forma precisa, o papel desempenhado por controle na especificação de algoritmos. Ele mostrou que, num certo formalismo ("programação

lógica"), é possível separar um algorítmo numa componente lógica e numa componente de controle. Kowalski mostrou que existem interpretadores "inteligentes" (mas, ineficientes) capazes de executar a lógica de um algorítmo com pouca ou quase nenhuma informação de controle. O controle requerido em programação convencional também pode ser usado em programação lógica; nesse caso, um interpretador "não tão inteligente" (mas, eficiente) é suficiente.

Neste trabalho procuramos explorar, esta separação entre lógica (fins) e controle (meios) em programação "data-flow" ("data-flow programming"). Uma significante classe de problemas de programação é naturalmente expressa em termos de "data-flow": a entrada e a saída são feixes de dados ("data streams"); tipicamente, o programa efetua uma transformação sobre o feixe de entrada ("input stream"). Um programa data-flow pode ser visto como um sistema de módulos, interconectados, com feixes de dados fluindo entre eles. A característica principal do modelo data-flow é que o sequenciamento das ativações/desativações dos vários módulos é controlado implícitamente pela maneira em que os módulos são conectados pelos feixes de dados. Especificamente, a ativação (ou desativação) de um módulo depende apenas da presença (ou ausência) de dados no seu feixe de entrada. Naturalmente, a fim de que esta forma implícita de controle seja efetuada automaticamente, necessário se faz o uso de uma linguagem especial, que chamamos de linguagem data-flow. Neste artigo, mostramos que o modelo data-flow é útil, mesmo no contexto de programação com uma linguagem de programação sequencial convencional. Nesse caso, embora (ainda) tenhamos que nos preocupar com os meios, podemos pelo menos "dividir para conquistar" ("divide-to-conquer"): primeiro concentramo-nos no fim e produzimos módulos funcionalmente semelhantes àqueles de um programa data-flow; então, concentramo-nos nos meios, isto é, programamos explicitamente o controle necessário para estabelecer corretamente a cooperação entre os módulos. Este controle, conforme veremos, é fácil de programar. Além disso, temos a vantagem de, pelo menos parcialmente, separar considerações de controle da lógica do programa.

2. O MODELO DATA-FLOW

Um programa data-flow consiste de módulos, cada um com zero ou mais feixes de entrada e zero ou mais feixes de saída. Um feixe pode ser, simultâneamente, entrada para um módulo e saída para outro. Nesse caso, dizemos que existe um elo ("data-link") dirigido de um módulo para outro. Estes elos constituem a única forma de interação entre módulos no modelo data-flow.

No método descrito neste artigo usamos uma linguagem de programação (de alto nível) convencional, ou seja, WATFIV [8]. Primeiro assumimos o modelo data-flow: escrevemos separadamente cada módulo que expressa apenas uma função entre seus feixes de entrada e de saída. Nestes módulos, construtos tipicamente convencionais tais como IF-THEN-ELSE e WHILE-DO são permitidos. As ca

racterísticas de data-flow são incorporadas nos módulos através de dois operadores: GET e PUT. Estes operadores definem operações sobre feixes da seguinte maneira: sejam GET_i e PUT_i operações sobre o feixe s_i num programa. A avaliação de $GET_i(x)$ resulta no valor TRUE no caso do feixe s_i estar não vazio e em FALSE caso contrário. No primeiro caso, tem-se como efeito colateral a atribuição a x do próximo elemento de s_i. No segundo caso, a chamada não terminará e o módulo chamante ficará bloqueado durante todo o tempo em que s_i permanecer vazio. A avaliação de $PUT_i(x)$ tem como único resultado a inclusão de x como último elemento de s_i. Via GET e PUT, cada módulo, respectivamente, pode assumir que o próximo elemento de um feixe de entrada não vazio está sempre disponível e que um elemento poderá sempre ser adicionado a um feixe de saída. Um feixe pode ser imaginado como uma fila sem limites. Para feixes de entrada finitos, um módulo pára após completar o processamento do seu último dado de entrada. Uma vez completos os módulos, os feixes são implementados através de definições, subprogramas, para os operadores PUT_i e GET_i. Além disso, pequenas modificações nos módulos, também se fazem necessárias.

A seguir demonstramos o método através de um exemplo simples de processamento de texto devido a Naur [9]. Nesta seção apresentamos apenas a componente lógica do algorítmo na forma de módulos data-flow. Nas seções seguintes, apresentamos a componente de controle a qual, através da implementação dos feixes, estabelecerá o controle requerido para interação entre os módulos.

2.1 O Exemplo

O problema é escrever um programa que lendo uma sequência de caracteres, S, imprime uma sequência de linhas, S'. A sequência de entrada S é tal que cada caracter ou é um BLANK (branco), um NLCR ("New Line and Carriage Return), ou um "caracter de palavra" (isto é, qualquer caracter que não seja BLANK, NLCR, ou "%", o qual marca o fim de S). Na sequência de saída, S', o número de linhas impressas deve ser o menor possível. Cada linha é uma sequência de "palavras" separadas, entre si, por um único branco e deve conter no máximo COMLIN (comprimento da linha) caracteres. Cada palavra é uma subsequência maximal de "caracteres de palavras". Requer-se ainda que a concatenação de sucessivas palavras contidas nas sucessivas linhas seja igual à sequência de sucessivos caracteres de palavras contidos em S.

A seguir construimos um programa data-flow com dois módulos: MOD1 e MOD2. A entrada para MOD1 é a sequência S. Sua função é "filtrar" palavras de S. Nesta tarefa, MOD1 inclui um branco delimitador entre cada duas palavras consecutivas. Esta sequência de palavras, saída de MOD1, constitui a entrada para MOD2 que, então, completa o processamento requerido. Ao conectar a saída de MOD1 à entrada de MOD2, estabelecemos um elo dirigido

de MOD1 para MOD2. Associado com este elo existe um feixe de dados sobre o qual as únicas operações possíveis são $PUT_1(x)$ (por MOD1) e $GET_1(x)$ (por MOD2).

(Nota 1: O feixe de dados associado com $PUT_1(x)$ e $GET_1(x)$ pode ser visto tanto como um feixe de palavras (caso em que x assumirá palavras com valores) quanto como um feixe de caracteres (caso em que x assumirá caracteres como valores). No primeiro caso, a tarefa de empacotar e desempacotar caracteres em palavras é feita em PUT_1 e GET_1, de modo que os módulos ficam mais simples. No segundo caso, esta tarefa é realizada nos módulos, de modo que a simplificação cai em PUT_1 e GET_1. Escolheremos a última alternativa, isto é, um feixe de caracteres. Achamos que "caracteres" sendo mais primitivos do que "palavras", são mais apropriados no sentido de uma eventual padronização na forma de interconexão entre módulos devendo, com isso, facilitar a ligação entre módulos já existentes e novos módulos.)

(Nota 2: Excetuando PUTs e GETs, todos os identificadores que aparecem num módulo são locais a ele.)

Os módulos, MOD1 e MOD2, com exceção dos PUTs e GETs são codificados em WATFIV e são apresentados nos Quadros 1 e 2, respectivamente.

```
C       DECLARAÇÕES
C       NOTEOI (NOT-END-OF-INPUT): INICIALIZADO COM .TRUE.
        WHILE (NOTEOI) DO
          NOTEOI = GET0(CAR)
          WHILE (NOTEOI.AND.(CAR.EQ.BLANK.OR.CAR.EQ.NLCR)) DO
            NOTEOI 1 GET0(CAR)
          END WHILE
C         NESTE PONTO, CAR OU É UM CARACTER DE PALAVRA OU %
          WHILE (NOTEOI.AND.CAR.NE.BLANK.AND.CAR.NE.NLCR) DO
            PUT1 (CAR)
            NOTEOI = GET0(CAR)
          END WHILE
          IF (NOTEOI) THEN DO
C           INSERE BRANCO NO FIM DA PALAVRA
            PUT1 (BLANK)
          END IF
        END WHILE
        STOP
        END
```

Quadro 1

```
C       DECLRAÇÕES
C       ARRAYS PALVRA E LINHA TEM, AMBOS, TAMANHO COMLIN.
C       PRESUPOE-SE QUE TODA PALAVRA NO TEXTO DE ENTRADA TEM
C       COMPRIMENTO MENOR OU IGUAL A COMLIN.
C       NOTEOI: INICIALIZAÇÃO COM .TRUE.
        INDLIN = 0
        WHILE (NOTEOI) DO
          COMP = 0
          NOTEOI = GET1(CAR)
          WHILE (NOTEOI .AND. CAR.NE.BLANK) DO
            COMP = COMP + 1
            PALVRA (COMP) = CAR
            NOTEOI = GET1 (CAR)
          END WHILE
          IF((INDLIN + COMP) .GT. COMLIN) THEN DO
            PRINT, (LINHA(I), I = 1, INDLIN)
            INDLIN = 0
          END IF
          I = 1
          WHILE (I .LE. COMP) DO
            INDLIN = INDLIN + 1
            LINHA (INDLIN) =  PALVRA(I)
            I = I + 1
          END WHILE
          IF (INDLIN .LT. COMLIN .AND. NOTEOI) THEN DO
            INDLIN = INDLIN + 1
            LINHA (INDLIN) = BLANK
          END IF
        END WHILE
        IF (INDLIN .GT. 0) THEN DO
          PRINT, (LINHA(I), I=1, INDLIN)
        END IF
        STOP
        END
```

Quadro 2

3. GERENCIAMENTO SISTEMÁTICO DE CONTROLE EM PROGRAMAS DATA-FLOW

O programa data-flow exemplificado neste artigo é "linear", isto é, o elo (único, no caso) existente entre os módulos impõe uma ordem linear entre os mesmos. Generalizando, dizemos que, um programa data-flow é linear, quando cada módulo tem no máximo um feixe de entrada e um feixe de saída.

Um feixe é, na realidade, uma fila com disciplina FIFO ("First-In-First-Out") sem dimensão prefixada. No modelo data -flow, qualquer subconjunto dos módulos, com velocidades relativas diferentes, pode ser simultâneamente ativado; a única restrição é que um módulo deve esperar quando de uma operação GET mal sucedi

da, isto é, sobre um feixe de entrada vazio. A seguir, introduzimos, sucessivamente, restrições adicionais necessárias no controle inter-módulos requerido por linguagens sequenciais convencionais.

R1. As filas tem ura dimensão limitada. Com isso, módulos também deverão esperar quando de uma operação PUT numa fila cheia.

R2. Em qualquer instante, apenas um módulo executa. A partir de agora, podemos dizer que um módulo "tem controle", ou não, e que um módulo "transfere controle" para outro módulo.

R3. Transferências de controle ocorrem apenas entre "vizinhos". Dois módulos são vizinhos quando existe um elo entre eles.

Das restrições acima segue-se que se módulo X transfere controle para módulo Y, então caso X seja reativado, esta reativação se dará devido a uma transferência de controle de Y para X. Além disso, esta transferência não deverá provocar uma nova ativação mas sim, um restabelecimento da última ativação. Isto significa que para cada módulo, em qualquer instante, existirá apenas uma ativação. Toda a informação de controle consistirá apenas de simples rótulos (marcando os pontos de retorno) em cada módulo. Este esquema de controle é consideravelmente mais simples que aquele usado com corotinas irrestritas, em que cada uma requer uma pilha de pontos de retorno. A próxima restrição, R4, estabelece uma estratégia de escalonamento ("scheduling strategy") que simplifica a implementação de filas.

R4. Cada módulo executa o máximo de tempo possível. Isto é, até que sua fila de entrada (saída) se torne vazia (cheia), o que ocorrer primeiro.

Esta restrição, implica na seguinte propriedade:

Propriedade 1: Quando um elemento é inserido numa fila, deleções não ocorrerão até que a fila se torne cheia (a não ser que a marca de fim de dados no feixe de entrada seja encontrada primeiro). Similarmente, quando um elemento é deletado duma fila, inserções não ocorrerão sem que a fila primeiro se esvazie.

Com esta propriedade, não precisamos mais de filas no sentido usual, o que não introduz quaisquer restrições no ordem de ocorrências de deleções e inserções. A partir de agora, nos referiremos a filas com "buffers".

Uma outra propriedade, agora decorrente do modelo data-flow, é o que chamamos de propriedade da "auto-suficiência":

Propriedade 2: O correto funcionamento de cada módulo ja

mais dependerá da presença simultânea de elementos num buffer. Noutras palavras, cada buffer poderá ter qualquer dimensão que seja pelo menos igual a dimensão de um elemento. Caso elementos seja necessários simultaneamente, o módulo em questão deverá prover armazenamento interno para tais.

4. UTILIZAÇÃO DAS RESTRIÇÕES R1, R2, R3 E R4.

Nesta seção descrevemos como usar as restrições R1, R2, R3 e R4 para completar a transformação dos módulos data-flow da Seção 2 num programaWATFIV. Conforme veremos, a modificação mais importante a ser feita nos módulos é a adição de subprogramas definindo as operações PUT e GET.

O i-ésimo módulo (i=2,...,n-1) de um programa data-flow (linear) obtém seus dados de entrada (via GET_{i-1}) do seu (i-1)-ésimo buffer e coloca sua saída (via PUT_i) no i-ésimo buffer. $Módulo_{i+1}$ é o "consumidor" de $módulo_i$; $módulo_i$ é o "produtor" do $módulo_{i+1}$. GET_i e PUT_i atuam sobre o mesmo $buffer_i$. Nem os módulos nem outros GETs e PUTs tem acesso do $buffer_i$. (Lembre que os módulos só tem acesso aos buffers indiretamente, via GETs e PUTs). $Módulo_1$ e $módulo_n$ são as interfaces do programa com o meio exterior. Assim sendo, GET_o e PUT_n são tratados separadamente

Quando $módulo_{i+1}$ requisita um dado de entrada, do ponto de vista de data-flow só existem duas possibilidades distinguíveis: ou o feixe de entrada é vazio ou não. Na realidade, existe ainda o caso em que embora o buffer esteja vazio o feixe de entrada ainda não se esgotou. Esta última distinção pode ser resolvida em GET_i e com isso $módulo_{i+1}$ permanece inalterado. Nesse caso, GET_i se encarregará de produzir o próximo elemento de entrada: se este estiver no buffer, GET_i tira-o de lá retornando-o para o $módulo_{i+1}$; caso contrário, GET_i primeiro ativa $módulo_i$ para encher o buffer. De maneira similar, a complicação de um buffer cheio pode ser resolvida em PUT_i: se $buffer_i$ não estiver cheio, então PUT_i insere-lhe o elemento imediatamente; caso contrário, PUT_i primeiro causará a reativação de $módulo_{i+1}$ para esvaziar o buffer.

Com WATFIV, o compartilhamento dos buffers será convenientemente implementado via áreas COMMON. O esquema (tipo corotinas) de transferência de controle intermódulos será simulado via o mecanismo CALL/RETURN e o GO TO computado. Nesta simulação deve-se levar em conta a assimetria (implicada pelo mecanismo CALL/RETURN) entre rotinas chamante e chamada. Isto é, toda instrução RETURN num subprograma chamado deverá ser precedido pelo registro do ponto de reativação ou seja, a próxima instrução

seguinte ao RETURN envolvido. Um GO TO no começo do subprograma se encarregará, quando o subprograma for chamado, de transferir o controle para o ponto de reativação previamente registrado (ou para o início do subprograma no caso de sua primeira chamada). Com isto, temos duas possibilidades a considerar:

(I) Produtores são ativados por CALLs, via GETs. RETURNs (simples) nestes GETs garantirão reativação correta dos consumidores (chamantes). RETURNs (aos GETs) nos produtores, todavia, deverão ser precedidos pelo registro do ponto de reativação tendo em vista a próxima vez que o produtor for reativado.

(II) Análogo a (I): substitua simultaneamente as palavras "produtores" por "consumidores", "consumidores" por "produtores", e "GETs" por "PUTs".

Uma outra escolha a ser feita é quanto à ativação inicial: quem deveria ser ativado primeiro, módulo$_n$ ou módulo$_1$? Por um lado, a ativação de módulo$_n$ causa uma "sucção" dos dados através dos módulos; por outro lado, a ativação de módulo$_1$ faz com que os dados sejam "empurrados" através dos módulos. Referir-nos-emos a esses casos como "modo sucção" e "modo empurrão", respectivamente.

Das quatro combinações de uma escolha entre as possibilidades (I) e (II) e uma escolha entre modo sucção e modo empurrão, apenas duas são viáveis: com (I), modo sucção deve ser escolhido, caso contrário todos os módulos não serão ativados com um único CALL. Análogamente, com (II), modo empurrão deve ser escolhido.

Em adição a todos as considerações de controle discutidas até agora, resta-nos decidir sobre o "critério de parada" do programa. Podemos escolher entre entrada vazia e saída completa. Escolheremos a primeira alternativa, entrada vazia (parece mais fácil de manipular num programa). Comparemos, agora, o modo sucção com o modo empurrão assumindo entrada vazia como critério de parada:

Modo empurrão: a execução de módulo$_1$ é interrompida por causa do fim da entrada. Desde que, nesse ponto, os buffers 1,...,n-1 ainda podem conter dados para processamento, reativações adicionais de módulo$_i$ (i=2,...,n) são necessárias.

Modo sucção: quando módulo$_1$ deteta o fim de entrada já existem chamadas ativas o módulo$_i$ (i=2,...,n), de tal forma que quando módulo$_n$ pára, buffer$_i$ (i=1,...,n) estão todos vazios.

Aparentemente, o modo sucção juntamente com a alternativa (I) é a combinação correta.

Neste ponto, procederemos à transformação dos módulos data-flow num programa WATFIV. Consideraremos dois casos: execução sequencial com e sem buffers.

4.1 Execução sequencial com buffers.

Os módulos (com exceção de módulo$_n$ que é especificado como programa principal - desde que estamos no modo sucção) são transformados em subprogramas função do tipo LOGICAL. O valor verdade de um módulo indicará se sua ativação produziu, ou não, pelo menos um elemento de dados de saída. Logo um resultado .FALSE. implica que o módulo já produziu toda sua saída em chamadas anteriores.

As operações PUT$_i$ e GET$_i$ (i=1,...,n-1) também são definidas como funções do tipo LOGICAL e têm a seguinte forma:

```
LOGICAL FUNCTION PUTi(x)
COMMON/FEIXEi/< atributos do FEIXEi>
<  outras declarações >
IF (< BUFFERi cheio > ) THEN DO
  PUTi = .FALSE.
ELSE DO
   < x torna-se último elemento de BUFFERi >
  PUTi = .TRUE.
END IF
RETURN
END
```

```
LOGICAL  FUNCTION  GETi(x)
COMMON/FEIXEi/< atributos do FEIXEi>
<  outras declarações >
IF (< BUFFERi vazio >)  THEN DO
   <ajusta ponteiros em BUFFERi>
   IF (.NOT. MODULOi) THEN DO
    GETi = .FALSE.
    RETURN
   END
END IF
<  x torna-se próximo elemento de BUFFERi >
GETi = .TRUE.
RETURN
END
```

Agora, de acordo com a combinação alternativa (I) e modo sucção, módulos consumidores são ativados via RETURNs e módulos produtores via CALLs. Isto implica não só que chamadas a GET num módulo não precisam ser modificadas, como também que algumas mo

dificações nos módulos em conexão com chamadas a PUT são necessárias: suponha que módulo$_i$ chama PUT$_i$ e este retorna .FALSE. como resultado. Então, após garantir que em sua próxima reativação sua primeira providência será concluir esta operação de saída mal sucedida, módulo$_i$ deveria retornar controle (via GET$_i$) ao consumidor módulo$_{i+1}$. Assuma ainda que existem m chamadas a PUT$_i$ no texto de módulo$_i$ e que $L_1,\ldots,L_{m+2}$ são rótulos não existentes em módulo$_i$. Seguem-se, então, as regras:

Em módulo$_i$, substitua

RG1: a k-ésima chamada a PUT$_i$(x)	por	L_k	IF(.NOT.PUTi(x)) THEN DO
			CHAVE = k
			RETURN
			END IF
			MODULOi = .TRUE

RG2: a instrução STOP	por	L_{m+2}	CHAVE = m + 2 RETURN

Além disso, imediatamente antes da primeira instrução executável de módulo$_i$,

RG3: insira	L_{m+1}	MODULOi = .FALSE. GOTO($L_1,\ldots,L_{m+2}$), CHAVE

onde CHAVE é uma variável do tipo INTEGER inicializada com m+1 em tempo de compilação.

O programa resultante da aplicação dessas regras aos módulos da Seção 2 é mostrado no Apêndice.

4.2 Execução sequencial sem buffers

Baseados no nosso exemplo, a obtenção de programas pelo método apresentado (até agora) caracteriza-se:(a) pela facilidade na obtenção dos módulos data-flow (a aproximação inicial) e (b) pela simplicidade e sistematização da transformação dos módulos data-flow em programas WATFIV.

Todavia, no contexto de programação sequencial, devido ao excessivo número de vezes em que dados são movidos através dos módulos, o uso de buffers como armazenamento intermediário tende a ser ineficiente. Observa-se, no nosso exemplo, que um caracter desde sua entrada até sua saída no programa, é movido cinco vezes!

A seguir usamos a propriedade da auto-suficiência dos módulos no sentido de obter um programa (sequencial) mais eficiente. Segundo aquela propriedade, o programa obtido anteriormente

deveria funcionar com um buffer de qualquer dimensão diferente de zero. Em particular, para buffers de dimensão um, a intermediação de GETs e PUTs é desnecessária. Nesse caso, o novo esquema de controle pode ser implementado modificando os módulos data-flow de acordo com as regras RG1', RG2', RG3' e RG4' dadas abaixo:

RG1'. Em módulo$_{i+1}$, substitua GET$_i$(x) por MODULOi(x).

RG2'. Em módulo$_i$, substitua PUT$_i$(x) por

```
                MODULOi = .TRUE.
                CHAVE = k
                RETURN
L_k             CONTINUE
```

RG3'. Imediatamente antes da primeira instrução executável de módulo$_i$, insira:

```
                INTEGER CHAVE/m+1/
                MODULOi = .FALSE.
                GO TO(L_1,...,L_m+2), CHAVE
L_m+1
```

Finalmente,

RG4'. Substitua, em módulo$_i$, STOP por

```
                CHAVE = m+2
L_m+2           RETURN
```

O resultado da aplicação dessas regras aos módulos da Seção 2, não é mostrado por limitações de espaço. O leitor, porém, fica convidado a checá-lo. (Note-se que, como antes, os módulos, com exceção de módulo$_n$, são transformados em subprogramas função do tipo LOGICAL).

5. CONCLUSÕES

A decomposição de um algorítmo em componentes lógica e de controle proposta por Kawalski [6] é expressada através de um exemplo simples de processamento de texto em que a componente lógica do algorítmo é especificada por módulos data-flow lineares e independentes. Partindo desta especificação, apresentamos um método através do qual restrições de controle são sistematicamente introduzidas, em duas diferentes maneiras, do modo a transformar os módulos em diferentes programas sequenciais, em WATFIV. Num caso, o programa resultante usa buffers para conectar os módulos e portanto é conceitualmente mais próximo do modelo data-flow. Os buffers reduzem a velocidade de execução do programa por causa dos múltiplos movimentos dos dados através dos módulos. Para módulos auto-suficientes, num contexto sequencial, os buffers não adicionam vantagem ao programa. Apesar disso, o ca-

ráter sistemático (e simples) na especificação de controle do programa é, a nosso ver, uma das vantagens do método. No outro caso, o programa resultante é obtido explorando-se a propriedade da auto-suficiência. Dados são transferidos diretamente entre os módulos eliminando-se, com isso o "overhead" no uso dos buffers. O programa é mais rápido, e até certo ponto, também aproxima-se do modelo data-flow.

No exemplo considerado, observamos ainda que o problema apresenta uma estrutura hierárquica: a saída requerida é uma sequência de linhas, cada uma das quais é uma sequência de palavras que, por sua vez, são sequências de caracteres. Comumente, problemas desse tipo são resolvidos com ninhos de laços("nested loops"): o laço mais externo produz as linhas ativando um laço interno que produz palavras processando caracteres. Um ponto a investigar é se, de alguma forma, módulos data-flow que manipulam dados hierárquicamente estruturados (como no nosso exemplo) podem ser combinados num programa monolítico similar à versão dos laços aninhados. Aparentemente, isto é possível quando os módulos são tais que as ocorrências de PUTs e GETs são limitadas a um certo número e certas posições dentro dos módulos. Para módulos arbitrários, todavia, a inter-relação dos dois modelos (data-flow e ninhos de laços) não parece óbvia.

Em continuidade ao trabalho, estamos agora desenvolvendo um "pré-processador data-flow" [7], cujo objetivo é implementar a transformação de módulos data-flow em programas WATFIV, não apenas de forma sistemática mas, também, automaticamente.

6. TRABALHOS RELACIONADOS

Há já algum tempo, trabalhos nas áreas de arquiteturas de de computadores e linguagens de programação vem sendo influenciados pelo modelo data-flow [2, 5]. Em [10], Peacock apresenta uma excelente "survey" de linguagens de programação ("hardware-orientadas") baseadas no modelo data-flow. Em [3], Kahn e McQueen propõem um atraente modelo computacional em que características específicas da arquitetura do processador são abstraídas da linguagem data-flow associada. Ainda nesta linha, apresentamos em [11] como usar o modelo data-flow no contexto de programação lógica. Na área de metodologia de programação, exemplos de trabalhos influenciados pelo modelo data-flow são encontrados em [1, 4].

7. REFERÊNCIAS

1. CUNHA, P.R.F. & MAIBAUM, T.S.E. - A Communications Data Type for Message Oriented Programming. Lecture Notes in Computer Science, Springer-Verlag, vol. 83, 1980.

2. DENNIS, J.B. - Programming Generality, Parallelism, and Computer Architecture. Proc. IFIP 68.

3. KAHN, G. & McQUEEN, D.B. - Coroutines and Networks of Parallel Processes. Proc. IFIP 77.

4. KERNIGHAN, B.W. & PLAUGER, P.J. - Software Tools. Addison-Wesley, 1976.

5. KOSINSKI, P.R., - A Data-Flow Programming Language. IBM Research Tech Report RC 4264, 1973.

6. KOWALSKI, R.A. - Algorithm = Logic + Control. Research Report, Dept. of Computation and Control, Imperial College, 1977.

7. LUCENA FILHO, G.J. & MENDES, A.M.M., - Um Preprocessador Data-Flow. (Em preparação).

8. MOORE, J.B. & MAKELA, L.J. - Structured FORTRAN with WATFIV. Reston Publishing Company, Inc., 1978

9. NAUR, P., - Programming by Action Clusters. BIT 9, 1966.

10. PEACOCK, J.K. - A Survey of Data-Flow Programming Languages. Master thesis, Dept. of Applied Analysis and Computer Science, Univ. of Waterloo, 1976.

11. Van EMDEN, M.H., LUCENA FILHO, G.J. & SILVA, H.M., - Predicate Logic as a Language for Parallel Programming. Submetido para publicaçao nas Proc. of the Logic Programming Workshop, Hungria, 1980.

```
$JOB GENTIL
$NOEXT
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     PROGRAMA PRINCIPAL: MODULO2
C     USA ROTINA 'INICIA' PARA INICIALIZAR FEIXES
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      CHARACTER PALVRA(60), LINHA(60), BLANK/' '/, CAR
      INTEGER INDLIN, COMP, COMLIN/60/, I
      LOGICAL GET1, NOTEOI/.TRUE./
C
      CALL INICIA
      INDLIN=0
      WHILE(NOTEOI) DO
        COMP=0
        NOTEOI=GET1(CAR)
        WHILE(NOTEOI .AND. CAR .NE. BLANK) DO
          COMP=COMP+1
          PALVRA (COMP)=CAR
          NOTEOI=GET1(CAR)
        END WHILE
        IF((INDLIN+COMP) .GT. COMLIN) THEN DO
          PRINT 10,(LINHA(I),I=1,INDLIN)
          INDLIN=0
        END IF
        I=1
        WHILE(I .LE. COMP) DO
          INDLIN=INDLIN+1
          LINHA(INDLIN)=PALVRA(I)
          I=I+1
        END WHILE
        IF(INDLIN .LT. COMLIN .AND. NOTEOI) THEN DO
          INDLIN=INDLIN+1
          LINHA(INDLIN)=BLANK
        END IF
      END WHILE
      IF(INDLIN .GT. 0) THEN DO
        PRINT 10,(LINHA(I),I=1,INDLIN)
      END IF
   10 FORMAT(1X,120A1)
      STOP
      END
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C                          MODULO1
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      LOGICAL FUNCTION MOD1(IDUMMY)
      CHARACTER CAR,NLCR/'@'/,BLANK/' '/
      INTEGER CHAVE/3/
      LOGICAL GET0, PUT1, NOTEOI/.TRUE./
C
      MOD1=.FALSE.
      GO TO (20,30,10,40), CHAVE
   10 WHILE(NOTEOI) DO
        NOTEOI=GET0(CAR)
        WHILE(NOTEOI .AND. (CAR .EQ. BLANK .OR. CAR .EQ. NLCR))  DO
```



```
          NOTEOI=GET0(CAR)
        END WHILE
C       NESTE PONTO CAR OU E UM CARACTER DE PALAVRA OU $
        WHILE(NETEOI .AND. CAR .NE. BLANK .AND. CAR .NE. NLCR) DO
   20     IF(.NOT. PUT1(CAR)) THEN DO
            CHAVE=1
            RETURN
          END IF
          MOD1=.TRUE.
          NOTEOI=GET0(CAR)
        END WHILE
        IF(NETEOI) THEN DO
C       INSERE BRANCO NO FIM DA PALAVRA
   30     IF(.NOT. PUT1(BLANK))THEN DO
            CHAVE=2
            RETURN
          END IF
          MOD1=.TRUE.
        END IF
      END WHILE
      CHAVE=4
   40 RETURN
      END
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL FUNCTION PUT1(CAR)
      COMMON/FEIXE1/BUF1(120),PRIM,ULT,TAMBUF
      CHARACTER BUF1,CAR
      INTEGER ULT,PRIM,TAMBUF
C
      IF((ULT+1).GT.TAMBUF) THEN DO
        PUT1=.FALSE.
      ELSE DO
        ULT=ULT+1
        BUF1(ULT)=CAR
        PUT1=.TRUE.
      END IF
      RETURN
      END
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL FUNCTION GET1(CAR)
      COMMON/ FEIXE1/BUF1(120),PRIM,ULT,TAMBUF
      CHARACTER BUF1,CAR
      INTEGER PRIM,ULT,TAMBUF
      LOGICAL MOD1
C
      IF(PRIM.GT.ULT) THEN DO
        ULT=0
        PRIM=1
        IF(.NOT.MOD1(0)) THEN DO
          GET1=.FALSE.
          RETURN
        END IF
      END IF
```

```
      CAR=BUF1(PRIM)
      PRIM=PRIM+1
      GET1=.TRUE.
      RETURN
      END
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      SUBROUTINE INICIA
      COMMON/FEIXE/BUF1(120),PRIM,ULT,TAMBUF
      INTEGER PRIM,ULT,TAMBUF
      CHARACTER BUF1
C
      READ,TAMBUF
      PRIM=TAMBUF
      ULT=0
      RETURN
      END
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      LOGICAL FUNCTION GET0(CAR)
      CHARACTER CARTAO(80),CAR,EOI/'%'/
      INTEGER I/80/
C
      GET0=.TRUE.
      IF(I.GE.80) THEN DO
        READ(5,10)CARTAO
        I=0
      END IF
      I=I+1
      CAR=CARTAO(I)
      IF(CAR.EQ.EOI) GET0=.FALSE.
   10 FORMAT(80A1)
      RETURN
      END
$ENTRY
```

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

CONSIDERACIONES INICIALES PARA EL DISEÑO DE UN NUEVO LENGUAJE DE PROGRAMACION

Javier De La Cuba Bravo
Central de Procesamiento de Datos
Ministerio de Marina
Av. La Marina Cuadra 36 s/n., Lima PERU.

El desarrollo de software en general, constituye una desafiante disciplina de las ciencias de computación que, desafortunadamente y pese a su gran trascendencia, es normalmente considerada como de poca importancia, permitiendo este hecho, que el desarrollo de proyectos de software se realize de manera bastante especial y por lo general, sin tener en cuenta experiencias anteriores ni recientes avances teóricos de la disciplina.

De igual manera, existe poca cooperación entre distintos grupos de desarrollo de software, para la realización del esfuerzo de diseño funcional del proyecto, dejándose esta importante etapa prácticamente en la imaginación, experiencia y "buen juicio" del equipo de programadores.

Reconsideremos por un momento los pasos básicos necesarios para lograr un desarrollo satisfactorio de software en general. Además de simplemente RESOLVER EL PROBLEMA (cualquiera sea la definición de este objetivo), se deben también considerar los siguientes aspectos:

1) Costos mínimos de desarrollo - vale decir, la mano de obra necesaria, tiempo de computador y otros recursos necesarios para detallar y escribir todo el software.

2) Costos mínimos de pruebas - tiempo de computación, esfuerzo, y otros recursos necesarios para probar "satisfactoriamente" dicho software.

3) Portabilidad - la habilidad para poder variar las condiciones ambientales de ejecución de dicho software, sin que este sufra grandes variaciones, o se vea afectado en su ejecución.

4) Mantenimiento - la habilidad para poder sostener dicho software en forma operativa durante un tiempo razonable mediante pequeñas modificaciones y sin encontrar mayores problemas.

5) Confiabilidad - La ausencia de errores graves y/o leves

6) Eficiencia - el permitir un uso eficiente de CPU, memoria, canales y otros recursos del sistema.

Es interesante anotar que es quizás la última característica, eficiencia, la que más atrae la atención del programador común.

Asumiendo que los objetivos son bien conocidos y aceptables para todo proyecto de desarrollo de software, es también significativo que no se haya desarrollado hasta la fecha, algún método de diseño que garantice la obtención final de todos estos requerimientos "simples".

Se puede observar asimismo, la importancia tan grande que representa la presencia de los lenguajes de programación de alto nivel en el proceso de desarrollo de software. Estos constituyen en sus diversas formas, importantes herramientas usadas en la proyección final del diseño de una aplicación, es decir, constituir un producto útil.

Pese a que el desarrollo de dichos lenguajes de programación ha sido bastante acelerado, y existe en la actualidad una gran diversidad y variedad de estos, se puede afirmar que quizás la única propiedad común y fundamental que se ha propagado a lo largo de toda la historia de computación, es la de que todos los lenguajes de programación permiten de alguna manera la representación de procedimientos y conceptos de tal forma que sean entendibles tanto por seres humanos como por un computador, luego de ser "traducidos" a un conjunto de instrucciones de máquina que al ser ejecutadas reflejen las intenciones de su creador.

Aunque hubiese sido quizás lo mas natural, que cada nueva generación de lenguajes de programación aprendiese de las anteriores de manera tal que se adaptasen a su correspondiente problemática de desarrollo, pareciera ser que esto no es lo que ha sucedido en la realidad. Aparentemente, no se tiene en cuenta el desarrollo de disciplinas tales como Ingeniería de Software, que han experimentado un acelerado y espectacular desarrollo en los últimos tiempos, pese a que como dice el Profesor Richard W. Hamming:

> "...existe una gran diferencia entre la Ingeniería como ciencia y la Ingeniería de Software, en que mien-

tras la primera se basa principalmente en reglas del mundo real, la segunda se desenvuelve siempre dentro de un ambiente producido por el hombre (MUNDO SINTETICO), el mismo que por definición, se halla sujeto a constantes variaciones..."

Esta misma problemática de desarrollo de software, permite considerar como factible el hecho de que se puede lograr un lenguaje de programación que considere como natural y propio de su estructura, el desarrollo de software utilizando para ello las mejores y más reconocidas técnicas de diseño y desarrollo, así como que permita la consecución final de los objetivos planteados anteriormente.

De igual manera, dicho lenguaje debiera constituir una imagen del proceso total del diseño utilizado, de tal forma que se convierta en la base común de los usuarios, analistas y programadores, así como que permita una reducción significativa de la dificultad existente actualmente tanto en la etapa de chequeo del producto, como de los problemas de inter-comunicación típicos de un grupo de desarrollo de software.

Es por consiguiente el propósito de este artículo, el presentar las consideraciones iniciales mínimas para el diseño de un lenguaje de programación que posea las características antes descritas, y que sirva de base para su desarrollo futuro; dicho lenguaje deberá tener una base sintáctica mínima que le permita una fácil y natural adaptación al ambiente en que se use, así como representar y constituir una real imagen de la aplicación desarrollada; esto implica necesariamente que dicho lenguaje deberá permitir la descripción funcional de un problema de tal manera, que facilite el desarrollo de software, a través de la utilización natural de los conceptos y principios de la Ingeniería de Software y abstracción funcional.

ANTECEDENTES Y BASE COMUN DE DISEÑO

Dado que no es del todo simple el poder separar de una manera completa los alcances, beneficios, desventajas e implicancias del binomio hardware-software en lo que a importancia y trascendencia se refiere, es necesario sin embargo, el establecer de una manera formal el rol histórico que dicho binomio ha jugado en el desarrollo de las ciencias de computación en general.

Históricamente, y sólo con el propósito de enfocar aquellos aspectos que tienen trascendencia en el desarrollo de este artículo, es necesario destacar la gran importancia que han tenido las características de hardware en general, en el desarrollo del software correspondiente, asi como señalar que han constituído un factor muy importante en la evolución de los distintos lenguajes de programación, al igual que en su tendencia futura.

Estos aspectos van desde el desarrollo propio de nuevos

elementos electrónicos y técnicas de miniaturización de circuitos, hasta el giro económico que éstas mismas han ocasionado. A nadie debe escapar que la tendencia actual de crecimiento de costos del software, ya sea por diseño o mantenimiento, continuará en una vertiginosa escalada, mientras que el costo del hardware, se verá cada vez más y más reducido. Esto ha ocasionado que el deseo de disminuir los costos del software, haya sido dirigido hacia el desarrollo de nuevos métodos y técnicas que faciliten su desarrollo y/o mantenimiento.

Ya que el objetivo es el conseguir que el software realice la tarea para la cual fue concebido, es necesario señalar entonces, que este objetivo implicará que dicha tarea sea realizable sin errores graves y por consiguiente, con una confiabilidad casi total.

Myers (2) nos presenta dos definiciones interesantes:

- Error de software:

  "Un error de software se halla presente cuando éste no hace lo que el usuario espera razonablemente que haga. Una falla de software es una ocurrencia de un error de software".

- Confiabilidad de software:

  "Es la probabilidad de que el software ejecutará durante un período particular de tiempo sin fallar, multiplicado por el factor del costo al usuario, ocasionado por cada falla (error de software) encontrada".

Del estudio realizado sobre fallas de software y de la gran complejidad que significa el chequeo consistente del funcionamiento de programas y sistemas, se puede inferir que tales fallas no son normalmente causadas por "desgaste", sino por errores en el diseño básico inicial, en la programación en sí o en posteriores modificaciones debidas a la obsolescencia del producto, o mejor dicho, a su incapacidad para adaptarse naturalmente a las necesidades impuestas por un ambiente dinámicamente cambiante.

Esta situación se presenta de manera gráfica en la Figura N° 1 en la que se puede apreciar claramente, cual es el comportamiento típico de un sistema a través del tiempo, en relación al número de fallas de software.

Con el propósito de reducir estos inconvenientes asi como conseguir un software generalmente más barato de diseñar y mentener, se han desarrollado una serie de técnicas nuevas tales como:

- Programación estructurada
- Diseño de arriba hacia abajo (TOP-DOWN)

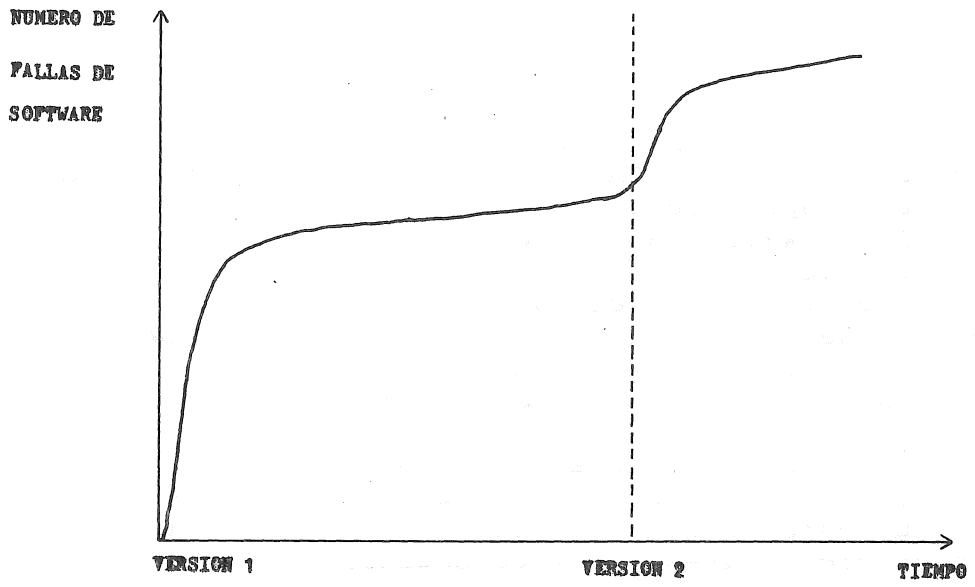


FIGURA Nº 1

- Diseño modular
- Desarrollo jerárquico
- etc.

Todas ellas tienen en común el hecho de qué concentran su atención en la determinación de límites acerca de la manera en la que se debe diseñar y programar software mediante la determinación de estructuras adecuadas; esto implica que en general, se debe diseñar y programar utilizando aspectos y métodos disponibles en el lenguaje escogido. Al realizar ésto, es posible que la integridad del diseño sea perdida, por lo que se añadiría - cierto grado de complejidad al problema.

Para obviar dicha complejidad, se ha desarrollado una técnica muy poderosa que consiste en utilizar una abstracción del problema; ésto implica que en lugar de tratar directamente con el problema, lo que se hace es tratar con un modelo idealizado del mismo, que sí sea manejable. Dicha abstracción puede ser realizada al nivel de diseño, o en los niveles de representación de datos o de las estructuras de control necesarias.

El inconveniente ocurre cuando, debido a las limitaciones naturales del lenguaje escogido para poder representar convenientemente dicha abstracción, se producen efectos laterales desconocidos y que pueden influenciar indirectamente el comportamiento de un programa, ocasionando la presencia de fallas que son muy difíciles de detectar.

Estas fallas sin embargo, no son producto del diseño original, sino mas bien, un resultado directo de la incapacidad del lenguaje de programación de permitir una transformación directa

del diseño, en programas que funcionen adecuadamente; el problema básico se halla entonces, en que en lugar de integrar los lenguajes en los nuevos métodos de diseño, lo que se ha hecho es "ajustar" estos lenguajes a dichas técnicas de diseño (i.e. programación estructurada usando FORTRAN!!)

Dos aspectos más acerca de abstracción. No se desea eliminar con la abstracción, todas aquellas construcciones oscuras y confusas que resultan inadecuadas para el programador por trabajar a un nivel muy cercano al del hardware, sino por el contrario, se trata de hacer que no "aparezcan" el nivel de programa fuente, que sí es el nivel de trabajo del Programador. Por otro lado, tal como señala W.A. Wulf (3), se debe notar que la noción de abstracción puede aparecer en un lenguaje de programación, en cualquiera de dos maneras: implícita o explícitamente. Como abstracciones implícitas, se entiende todas aquellas impuestas por el lenguaje, tales como: estructuras de datos predefinidas (variables simples, registros, arreglos, etc.), estructuras de control predefinidas (DO WHILE, DO UNTIL, FOR, etc.) y estrategias de definición de la realización de dichas estructuras de datos y de control (código generado, administración, de memoria, etc). Como abstracciones explícitas, se entiende todas aquellas producidas e introducidas por el Programador, y que son soportadas por el lenguaje en un nivel casi metalingüístico, en el sentido de que provee de los mecanismos necesarios para la definición de dichas abstracciones (PROCEDURE, FUNCTION, nuevas definiciones de tipos de datos, etc.).

Pareciéra ser sin embargo, que el problema se halla precisamente en que los lenguajes contienen demasiadas abstracciones realizadas de manera implícita, lo que motiva que gran parte del esfuerzo de diseño se vea dirigido al "encapsulamiento" del problema en el esquema más o menos rígido que es planteado por el lenguaje; ésto es especialmente cierto si se trata de utilizar construcciones, o mejor dicho abstracciones que se hallen por debajo del ya "alto nivel" proporcionado por el lenguaje(i.e. trabajar con variables tipo puntero en COBOL).

Lo contrario es también cierto, en el sentido de que normalmente no se cuenta con suficientes mecanismos como para poder definir y utilizar abstracciones del tipo explícito, lo que realmente establece el límite superior del ambiente de desarrollo de software, al utilizar dichos lenguajes.

Un aspecto muy importante de la abstracción, la constituye el uso de datos abstractos. Bárbara Liskov y Stephen Zilles (4) nos proveen de una acertada definición de data abstracta:

"Un grupo de funciones u operaciones relacionadas que actúan sobre una clase particular de objetos, con la restricción de que el comportamiento de los objetos puede ser observado sólo mediante la aplicación de dichas operaciones".

La forma más común de data abstracta, la constituye la definición de tipo de datos, cuyas propiedades pueden ser descritas formalmente, facilitando la construcción de pruebas formales

o informales de la bondad de un programa; cuando un programa se escribe usando sólo construcciones primitivas de un tipo de dato abstracto, su realización al nivel más bajo puede ser variada sustancialmente, sin afectar la operación del programa.

En adición al concepto de data abstracta, se debe tener en consideración otros dos principios muy importantes, siendo el primero de ellos el de ocultamiento de la información, el mismo que consiste básicamente en no sólo hacer visibles ciertas propiedades esenciales de un módulo de software, sino en hacer inaccesibles al Programador aquellos detalles que no tengan importancia. El segundo principio es el de localización, que puede ser aplicado tanto a datos como a estructuras de control resultando en la conocida programación estructurada.

En este punto estamos ya en condiciones de presentar las consideraciones mínimas para el diseño de un lenguaje de programación basado en los aspectos discutidos; es necesario indicar que no se pretende presentar detalles de implementación. Todo lo que debe hacerse, es suponer que dicho lenguaje puede usarse para desarrollar programas "abstractos de alto nivel" que representarán una imagen del proceso que pretenden ejecutar, así como que existe algún medio (compilador) para poder traducirlo a una versión ejecutable en cierto hardware. La representación gráfica utilizada, así como el esquema general se hallan basados en el trabajo de J.Paccassi y C.Wick (1).

DISEÑO INICIAL

El componente principal de este lenguaje es lo que denominaremos PROCESO. Un PROCESO es la menor unidad sintáctica capaz de ser traducida a código ejecutable, por lo que debe ser lo suficientemente completa como para desarrollar una acción. Es lo que más se parece a un programa tradicional. La definición de PROCESO es totalmente recursiva, por lo que un PROCESO puede estar compuesto por sí sólo, o ser utilizado como una subunidad sintáctica de otros PROCESOS exteriores a él; en todo caso, un PROCESO es capaz de existir por sí sólo, requeriendo únicamente del ambiente operativo en que se ejecute. Esto no quiere decir que necesariamente se resuelva todas sus referencias externas a tiempo de traducción, sino que simplemente al momento de ser compilado es transformado a un nivel tal que le sea permitido ejecutar en el ambiente de operación seleccionado; estos detalles corresponden al modo de implementación que se decida utilizar, por lo que deben permanecer ignorados por el momento.

La correcta definición de un PROCESO en sus aspectos sintácticos y semánticos, se basa en la adecuada secuencia de aquellos elementos lexicográficos que contenga; dichos elementos pueden clasificarse en general en dos categorías: TOKENS y SEPARADORES.

Un TOKEN representa una unidad lexicográfica, y constituye la esencia de los elementos del lenguaje. Ejemplos típicos

de TOKEN son: nombres, palabras reservadas, literales, símbolos especiales, etc.; asimismo, un TOKEN puede estar compuesto por uno o más caracteres, que dependerán del tipo de código que se utilice (EBCDIC, ASCII, etc.).

Un SEPARADOR que como su nombre lo indica, cumple la función de proveer cierta "distancia" entre TOKENS, la misma que es necesaria cuando la yuxtaposición de dos TOKENS pudiera hacer que aparezcan como uno sólo para el traductor. Ejemplos pueden ser: espacios en blanco, comentarios, etc.; de igual manera, un SEPARADOR puede estar compuesto de uno o más caracteres.

Un PROCESO realiza sus funciones a través de operaciones en datos que pueden ser externos o internos al mismo; la figura número 2 muestra una representación abstracta de esta estructura.

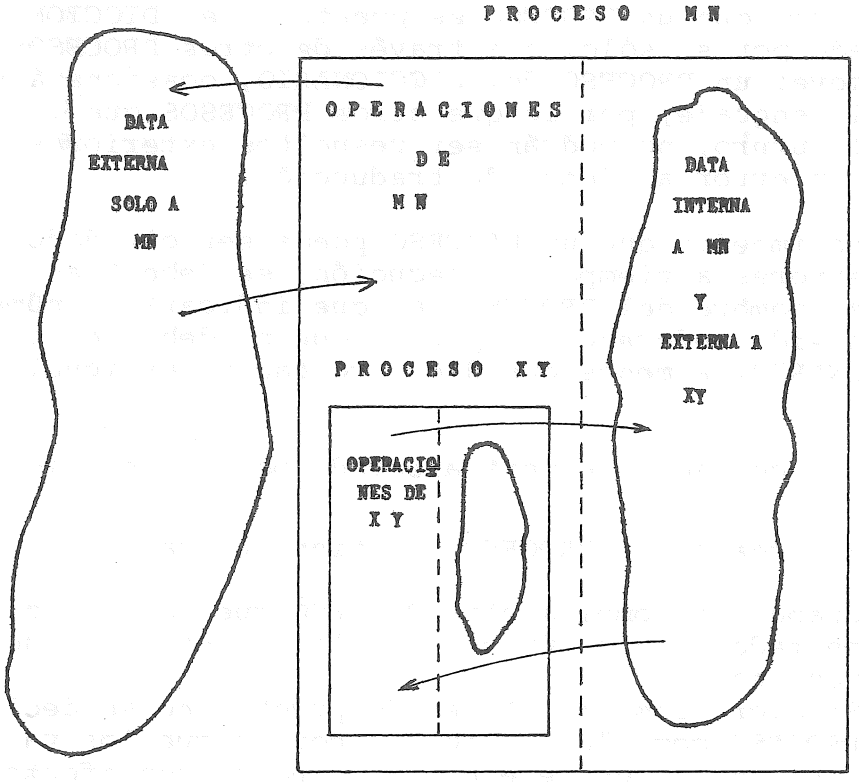


FIGURA Nº 2

Como se puede apreciar, el PROCESO XY existe por sí solo, pero a la vez forma parte del PROCESO MN; la data interna de MN, es a la vez externa para XY, mientras que la data externa de MN, "no existe" para XY, por no haberse definido dentro del alcance del mismo.

Todo PROCESO asimismo, tiene un nombre que lo hace único; sus componentes son:

- Interface de entrada

- Definición de datos
- Operaciones
- Interface de salida.

El único requisito de un PROCESO, deberá ser su nombre, pudiendo por lo tanto, contener ningún componente; esto es lo que denominaremos como PROCESO NULO.

El lenguaje es establecido a través del DICCIONARIO, que es alguna estructura de datos (posiblemente un archivo secuencial por índices, o una base de datos), cuya función es la de almacenar todos los PROCESOS que hayan sido definidos como "recordables"; dicho DICCIONARIO deberá proveer a su vez, de algún método para "olvidar" PROCESOS, asi como posiblemente algún tipo de operación entre PROCESOS, que facilite su ejecución en una forma pseudo-secuencial.

Una vez que un PROCESO es puesto en el DICCIONARIO, puede ser usado por sí sólo, o a través de otros PROCESOS; asimismo, el remover un PROCESO del DICCIONARIO, ocasionará su pérdida para el lenguaje, por lo que otros PROCESOS que lo referencien en el futuro, no podrán ser resueltos exteriormente, y se producirá un error a tiempo de traducción.

Para impedir que un PROCESO pueda ser olvidado y que esto cause errores a tiempo de ejecución, se deberá asociar un contador al nombre del PROCESO, el que indicará el número de veces que ha sido referenciado por lo que no debe ser removido del DICCIONARIO, a menos que dicho contador sea igual a zero binario.

Dicho contador se verá afectado por las siguientes acciones:

- Al crearse un PROCESO, contiene un valor de zero binario.
- Cuando se compila otro PROCESO que lo utiliza, es aumentado en uno, únicamente la primera vez que es referenciado.
- Si dicha compilación es de prueba, no se declara el PROCESO como "aprendible", por lo que los valores de los contadores referenciados no se ven afectados; al"olvidarse" un PROCESO, los contadores respectivos se verán disminuídos en uno.

Como se puede apreciar, el mantenimiento del DICCIONARIO es realizado únicamente por el traductor respectivo; asimismo, si se considera la propiedad de que cada DICCIONARIO es único, y que contiene a su vez datos sobre la frecuencia de utilización de todos los PROCESOS definidos como "recordables", entonces es posible extraer de él, estadísticas de uso de los diferentes PROCESOS, así como otros datos útiles que pudieran considerarse durante la etapa de implementación del lenguaje.

INTERFACES DE ENTRADA Y DE SALIDA

El propósito de definir interfaces de entrada y salida, es el de proveer un medio o canal de transmisión de información entre procesos. La idea, es evitar que pudiesen producirse efectos indirectos no deseados debido a errores en el manejo de los datos internos y que afecten a datos externos no definidos para el PROCESO que causó el error. Las interfaces de entrada y salida son los únicos puntos a través de los cuales un PROCESO puede comunicarse con datos externos, al permitir la definición de todos los parámetros léxicos necesarios para hacer que un dato externo sea accesible al PROCESO a través de la interface de entrada o de permitir que datos externos puedan ser alterados mediante la manipulación adecuada de datos internos, a través de la interface de salida.

La interface de entrada debe constituir el único punto a través del cual se puede recibir o definir datos externos al PROCESO; si es que el PROCESO no requiriese de datos externos, entonces la interface de entrada será nula y podrá ser omitida.

La interface de salida constituirá el único punto a través del cual se podrá hacer que datos internos del PROCESO puedan ser compartidos. Se utiliza para definir el canal de comunicación de salida entre PROCESOS, lo que significa que aquellos datos internos que se vean transformados en datos externos a través de su definición en la interface de salida, podrían a su vez ser usados como datos externos por otros PROCESOS. De igual manera pudiera no ser necesario definir datos externos, en cuyo caso la interface de salida será nula y podrá ser omitida.

Esta posibilidad de compartir dinámica y simultáneamente datos externos a tiempo de ejecución, podría ocasionar serios problemas de transmisión de valores así como de alcance, por lo que se considera que debe ser más estudiada, antes de proceder con su implantación.

DESCRIPCION DE DATOS

Esta sección corresponde a la definición de todos los datos internos de un proceso; dado que deseamos brindar las mayores facilidades en cuanto al manejo de datos se refiere, es necesario distinguir entre dos aspectos muy importantes y que son frecuentemente confundidos: el concepto abstracto de una estructura de datos y una posible realización del mismo; mientras el primero tiene que ver con la definición formal del dato, el segundo se refiere a la manera en que se implementa, y a los posibles estados que puede asumir. El lenguaje propuesto debe proveer facilidades para ambos, y especialmente para producir diferentes realizaciones del mismo concepto. Nótese asimismo, que parte muy importante de la definición, es el posible uso que se le puede dar mediante la aplicación de ciertos Operadores tales como adición, multiplicación, intersección, etc.

Veamos, por claridad, un ejemplo de esta situación: asumamos que nuestro lenguaje permite la definición de un tipo de variables que se "comportan" como stacks; lo único que se nece-

sitaría para poder usarlos sería una expresión tal como:

DECLARE A,B: STACK

En donde se está declarando dos variables del tipo stack, llamadas A y B; a su vez, al definirse el tipo STACK, se deberá tener en cuenta las operaciones que se pueden realizar con él, su alcance, su nivel de protección, etc. Cuando un Programador define una variable como de tipo STACK, automáticamente asume todas sus propiedades, por lo que él sólo tiene que ver con operaciones primitivas tales como PUSH, POP, chequear si el stack está vacío, límites en su uso, etc. Asimismo, la realización de un stack en una máquina determinada, debe ser totalmente transparente al programador.

Esto sugiere que es deseable tener un cierto número de alternativas de implementación para todos los tipos de estructuras de datos que se definan, y que se pueda escojer aquella alternativa que sea más conveniente para determinado programa; sin embargo, es necesario incidir en el hecho de que el Programador no debe asumir ningún tipo de representación, por lo que él sólo debe utilizar aquellos primitivos que se definan para cada tipo de variable. Nuevamente, el problema de decidir cuál es el mejor método, estará a cargo del compilador respectivo, el mismo que seleccionará la mejor alternativa de entre un grupo de diferentes modos de implementación, que también podrían encontrarse definidos en el DICCIONARIO; esto implica que un PROCESO podría contener únicamente el componente de declaración de datos, y ser introducido en el DICCIONARIO para su posterior utilización.

Algunos aspectos que consideramos deben ser evaluados en la definición de datos, son:

- Nombre
- Propiedades de la estructura, las mismas que pueden ser "Importadas" (es decir, definidas con anterioridad en el DICCIONARIO).
- Elementos básicos de datos, (Ejm. bytes, apuntadores, bits, etc.).
- Tamaño de los datos (su representación en hardware, que podría ser dinámica).
- Información de valores iniciales
- Manejo de condiciones de error (interrupciones), por violación de las características de la estructura(OVERFLOW, UNDERFLOW, etc.).

OPERACIONES

El componente de operaciones constituye la parte "activa" de un PROCESO, en el que se realiza alguna tarea, ya sea utilizando las construcciones básicas (estructuras de control) del lenguaje, o mediante el uso de otros PROCESOS que se hallan definidos en el DICCIONARIO.

En este componente asimismo, se crea, modifica o destruye

datos locales, así como se utiliza las interfaces de entrada y salida. Un PROCESO asimismo, puede contener únicamente su componente de operaciones, en cuyo caso se trataría de la definición de algún tipo nuevo de estructura de control, que pueda ser utilizado por otros PROCESOS; este caso determina lo que denominaremos un PROCESO PURO.

Asi como se ha insistido mucho acerca de la importancia de la abstracción en la definición de datos, es necesario indicar que dicha abstracción es también posible en cuanto se refiere a estructuras de control, de asignación, y condicionales. Y esto se deduce fácilmente al coincidir en que si es posible abstraer datos y operaciones, entonces, es posible la abstracción en cuanto a la forma en que deben ser procesados.

Esto se debe precisamente a que al referirse de alguna manera a algún dato, se asocian su representación en el hardware, con la manera en que se comportan las estructuras de control que lo utilizan.

Así por ejemplo, todas aquellas estructuras de control que se relacionan con arreglos, deben necesariamente considerar como es que se hallan dispuesto en memoria (i.e. la suma de matrices en BASIC); sin embargo, dependiendo del Hardware que se utilice, el modo de secuenciamiento que se use será muy diferente si es que se trata de una máquina con paginamiento, o si el arreglo se ha representado como una lista concatenada.

De igual manera, la determinación de parámetros de estructuras de control, tales como el valor inicial, valor final y valor de incremento de una proposición FOR, dictan la manera en que éste se comporta sin tener en cuenta la representación de los datos en memoria, lo cual puede tener un costo muy alto.

El lenguaje propuesto deberá pues, contener alguna forma de poder expresar estructuras de control (o PROCESOS PUROS) que sean consecuencia de la adecuada utilización de los elementos básicos del lenguaje, así como deberá permitir una forma natural de escribir programas, que permitan la mejor representación de la tarea que se desea realizar. Para ello es necesario tener en cuenta que la abstracción en estructuras de control, no depende únicamente del tipo de acciones que se desea efectuar, sino también de la manera en que la definición de datos pudiese determinar cuan abstracto se puede ser en su utilización.

CONCLUSIONES

A través de una breve introducción a la problemática de desarrollo de software, se aprecia claramente que el enfoque tradicional utilizado se halla determinado en un gran porcentaje, por la posible aplicabilidad que pudiese tener determinado lenguaje de programación, y de su "armonía" con las nuevas técnicas de desarrollo de aplicaciones.

Asimismo, se ha examinado la posibilidad de usar técni-

cas de abstracción, que incluso puedan influir en el diseño mismo del lenguaje; se observa también, que la inclusión o no de determinadas estructuras en un lenguaje, tales como GO TO, CASE, DO ...WHILE, no afecta en general el tipo de aplicaciones que se puede realizar, sino únicamente el modo en que son desarrolladas. Y este es precisamente el punto que se desea hacer destacar; el proceso de desarrollo de software debe ser natural, no forzado ni tampoco limitado por restricciones "artificiales" que el mismo diseñador establece de manera indirecta; el diseño de un lenguaje debe por consiguiente, considerar el poder proveer por lo menos, suficientes facilidades que permitan una adecuada representación de algoritmos, y que sugiera mejores formas de poder expresarlos.

De igual manera se ha presentado en forma muy general, y sin pretender cubrir todos los detalles, aquellos aspectos que creemos debiera considerarse en el diseño de un nuevo lenguaje de programación; al haberse establecido únicamente un primer esqueleto de su estructura, se ha querido dejar para posterior estudio, todo lo concerniente al diseño final y a la etapa de implementación. Creemos sin embargo, que no existe el "lenguaje perfecto", ya que por más elaborado que este sea, no podrá impedir que se escriban con él, programas confusos y con errores, puesto que un programador ingenioso siempre encontrará un método para lograr confundirnos (es decir " ¿Porqué hacer las cosas de una manera fácil, si pueden ser difíciles?").

AGRADECIMIENTO

El autor desea expresar su agradecimiento a Jerry PACCASSI y Carl WICK, con quienes como compañeros de estudios, tuvo oportunidad de aprender mucho de su conversación; de igual manera, a Luis GUILLEN por prestarse voluntariamente a leer y comentar este trabajo; asimismo, al OM1. Juan PEZO y al OM2. Marceliano QUISPE, por su trabajo de mecanografiado.

Finalmente, pero ciertamente no en último lugar, a mi esposa Pelusa por su paciencia y compañía.

REFERENCIAS

1.- Jerry G. Paccassi y Carl C. WICK.- "A design for a Funcion - Descriptive Programming Language".- trabajo de tésis, USNPGS, 1978.

2.- Glenford J. Myers.- "Software Reliability. Principles and Practices".- John Wiley and Sons, Ing.

3.- William A. Wulf. "Languages and Structured Programs".- Prentice Hall, 1977.

4.- Bárbara Liskov y Stephen Zilles.- "An Introduction to Formal Specifications of Data Abstractions".- Prentice Hall, 1977.

5.- Edsger W. Dijkstra.- "Guarded Commands, Nondeterminacy and Formal Derivation of Programs".- Prentice Hall, 1977.

6.- James R. Low.- "Automatic Data Structure Selection: An Example and Overview".- Communications of the ACM, May 1978, Volume 21, Number 5, pp 376 - 384.

7.- James B. Morris.- " Data Abstraction: A Static Implementation Strategy".- Proceedings of the SIGPLAN Symposium on Compiler Construction, Denver, Colorado. August 6-10, 1979.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

EXPERIENCIAS SOBRE UNA METODOLOGIA DE PROGRAMACION

Rafael O. Fontao
Laboratorio de Sistemas Digitales
Departamento de Ingeniería
Universidad Nacional del Sur
Bahía Blanca - ARGENTINA

Juan A. Codagnone
Departamento de Sistemas
ATEC S.A. de Asesoramiento Técnico
Buenos Aires - ARGENTINA

INTRODUCCION

En este trabajo se expresan algunas de las ideas y experiencias en el uso de una metodología de programación. Esta técnica permite hacer frente a la crisis del ambiente informático, caracterizada especialmente por una indisciplina generalizada en el diseño e implementación de programas y sistemas.

Al igual que las normas de escritura de la programación estructurada clásica, esta metodología comparte sus objetivos. Sin embargo, como única estructura de control, y por lo tanto de pensamiento o concepción de programas, ofrece el autómata finito.

La metodología resultante permite unificar el diseño de programas para hacerlo independiente del lenguaje de programación. El trabajo consta de 4 secciones principales, la primera describe brevemente la metodología; las dos secciones siguientes tratan sobre las experiencias académica y profesional adquiridas con esta técnica. Finalmente la 4° sección propone las extensiones y líneas de trabajo a seguir por esta metodología.

DESCRIPCION DE LA METODOLOGIA

En un trabajo previo [2] fue presentada formalmente la metodología SOL. Esta metodología está soportada por un lenguaje cuyas características se incluyen resumidas en el apéndice.

La idea básica de esta metodología consiste en concebir a un programa como un autómata finito y determinista. Luego en sucesivos pasos, denominados refinamientos, se conciben los estados como otros autómatas a niveles inferiores.Este procedimiento concluye cuando todos los estados del autómata así concebido pueden sintetizarse mediante instrucciones en un lenguaje estandard disponible.

Un autómata finito (AF) modela un comportamiento secuencial que puede expresarse por una tabla de transiciones o su grafo equivalente. Partiendo de un estado inicial, el AF permanece en cada instante en un único estado produciendo una salida y ante el estímulo de una entrada, también finita, transita a un nuevo estado (eventualmente el mismo).

Así, se alternan sucesivamente salidas del AF con entradas al mismo hasta que arriba a un estado final y se detiene. La concepción de un AF se realiza en forma descendente expresando en pocos estados la descripción de su comportamiento. Sucesivamente los estados se van refinando en nuevos autómatas hasta alcanzar un nivel de detalle que puede implementarse con los recursos desponibles.

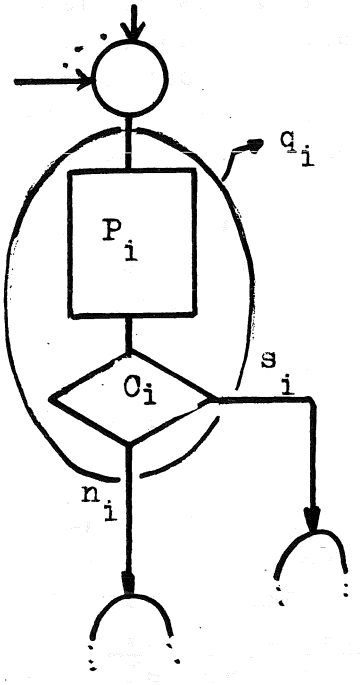
Análogamente, un programa o más generalmente la descripción de una tarea, puede concebírsela como un AF, utilizando para ello la noción clave que relaciona ambas disciplinas: interpretar el estado de un AF como tarea elemental de un programa.Por tarea elemental se entiende un paso del programa o tarea con un único punto de entrada y uno o más de salida. Similarmente el estado de un autómata es único a cada instante pero puede transitar a uno entre varios próximos estados posibles.

La salida de un estado será considerada como la acción de la tarea (no como el resultado de esa acción). Así por ejemplo, A= B+C como tarea es siempre la misma aún cuando

el resultado sobre A sea diferente al ajecutar B+C con otros valores. La acción de cada estado, a su vez, puede descomponerse en dos partes: la ejecución de la tarea y la evaluación de la próxima entrada.

La ejecución de la tarea corresponde a la aplicación de las herramientas de trabajo mientras que la evaluación de la próxima entrada (decisión) corresponde a la aplicación de herramientas de medida.(Las herramientas de trabajo transforman la materia prima:información, y las de medida permiten medir las propiedades de la materia prima) [2].

La Figura 1 muestra genéricamente una tabla de transiciones de un AF que modela un programa. A los efectos de simplificar el modelo sin perder generalidad cada estado tiene a lo sumo dos próximos estados posibles, esto es, basta una condición lógica para determinarlo.



	ENTRADA		ACCION	
	SI	NO	EJECUCION TAREA	EVALUACION PMA.ENTRADA
	.	.	.	.
	.	.	.	.
	.	.	.	.
	.	.	.	.
	.	.	.	.
	.	.	.	.
q_i	s_i	n_i	P_i	C_i
	.	.	.	.
	.	.	.	.
	.	.	.	.
	.	.	.	.

Fig. 1

Desde un punto de vista temporal, la evolución del AF es la siguiente:

ENTRADAS:		E^1	$E^2 \ldots E^t$	$E^{t+1} \ldots E^n$
ESTADOS:	$q_{inicial}$	q^1	$q^2 \ldots q^t = q_i$	$q^{t+1} \ldots q_{final}$
SALIDAS:		P^1	$P^2 \ldots P^t = P_i$	$P^{t+1} \ldots$ STOP

La evaluación, para cada instante t, de la próxima entrada es,

$$E^{t+1} \begin{cases} \text{SI} & : \text{ si } C_i \text{ es verdadera (o vacía)} \\ \text{NO} & : \text{ si } C_i \text{ es falsa} \end{cases}$$

por consiguiente el próximo estado, será :

$$q^{t+1} \begin{cases} s_i & : \text{ si } E^{t+1} \text{ es SI} \\ n_i & : \text{ si } E^{t+1} \text{ es NO} \end{cases}$$

La metodología SOL puede aplicarse en sucesivos pasos o refinamientos. A cada paso un módulo o parte del programa se refina explicitando en mayor detalle su comportamiento o programándolo directamente en el lenguaje de programación. Este proceso de refinamiento continúa hasta que todo el programa queda escrito en el lenguaje de programación.

De igual modo, un AF puede diseñarse en sucesivos pasos dando lugar a la siguiente metodología de programación:

Sea LPD el Lenguaje de Programación Disponible con el cual se implementará finalmente el programa.

Paso 0: Elija un lenguaje adecuado L para expresar sus ideas (idealmente un lenguaje natural) y conciba a todo el programa como si fuera un AF de un sólo estado.

Paso 1 a n: Si hay algún estado que no puede escribirse "claramente" en el LPD, entonces conciba a este estado (en L) como un AF. Esto es, un estado del AF es reemplazado por un conjunto de nuevos estados formando entre ellos un AF. (Como regla de claridad conciba cada AF con el menor número de estados posibles y teniendo en cuenta, a su vez, la validez de su concepción).

Paso n+1: (En este paso todos los estados de AF deberían ser "claramente" expresados en el LPD). Escriba en el LPD todos los estados del AF.

En esta metodología de programación, la estructura de control de un programa (Refinamiento, Pasos 1 a n) se escri-

be separadamente del resto de las sentencias (paso n+1). La estructura de control, a su vez, se modela por las transiciones de estado de un AF cuyos estados son el producto de un proceso de refinamientos. Es decir, cada estado cuya acción se describe en lenguaje natural, o se lo escribe en el LPD (paso n+1) si fuera simple o se lo concibe como un nuevo AF (paso entre 1 y n).

Por consiguiente, un programa puede visualizarse como una jerarquía de autómatas donde la relación de dependencia se describe por una estructura de árbol. La raíz de este árbol es un AF de un sólo estado (paso 0) que simboliza a todo el programa. Por cada refinamiento de estado se agregan al nodo del árbol correspondiente tantos descendientes directos como estados tenga el refinamiento. Este proceso continúa hasta que todo nodo terminal del árbol (o estado del AF) pueda modelarse con una acción escrita en el LPD. Ver a continuación.

```
REF
 1 DO...
   THEN ..
 2 DO ...
   THEN ..
 3 DO ...
   THEN ..
END

REF 2
 1 DO ...
   THEN ..
 2 DO ...
   THEN ..
END 2

REF 3
 1 DO ...
   THEN ..
 2 DO ...
   THEN ..
 3 DO ...
   THEN ..
END 3

REF 3.1
 1 DO ...
   THEN ..
 2 DO ...
   THEN ..
END 3.1
FIN
```

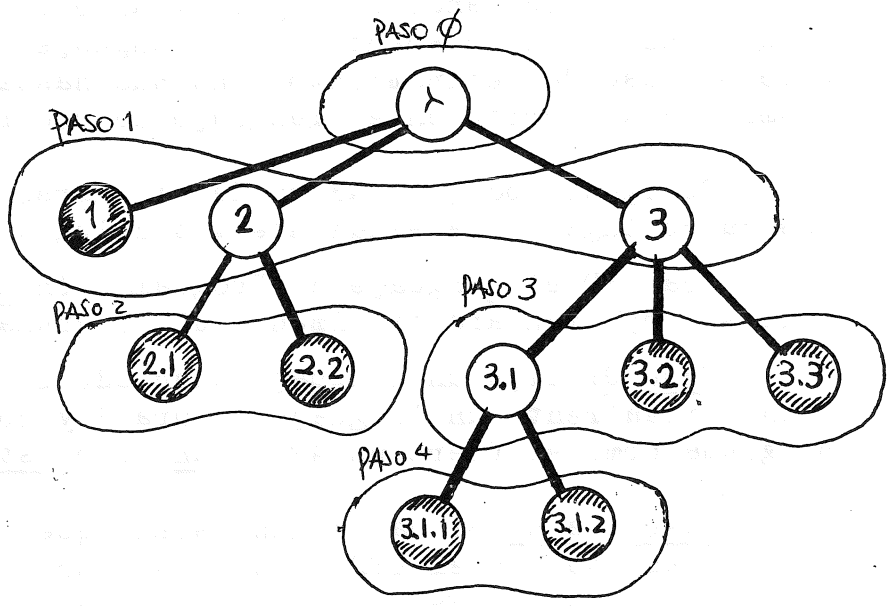


Fig. 2

A partir de aqui se escriben en LPD las hojas u nodos terminales del árbol que constituyen las acciones (nodos en sombra).

EXPERIENCIA ACADEMICA

La experiencia académica en programación suele caracterizarse por su influencia en la enseñanza y en el desarrollo de programas, sin sufrir necesariamente presión de factores externos, como pueden afectar a la experiencia profesional.

Esta particularidad es atractiva en cuanto el docente puede ensayar las metodologías de programación que con libertad académica juzque convenientes. Sin embargo, es en cierta medida la experiencia profesional, la que confirma la aplicabilidad de toda metodología.

En este sentido, la experiencia académica comenzó con el diseño y construcción de un precompilador SOL-BASIC [1] implementado para una minicomputadora PDP/8e de nuestra Universidad.

Poco tiempo después, razones accidentales motivaron dejar de usar el equipo mencionado con la configuración original. No obstante, la metodología resultante de programas en SOL, continuó su influencia en el diseño de programas y en la enseñanza misma de programación.

Cuando no se dispone del precompilador la aplicación de SOL consiste en expresar los refinamientos como si fueran comentarios del lenguaje disponible. Las acciones, por su parte, se escriben teniendo en cuenta la estructura de control establecido en los refinamientos, y en tal sentido las instrucciones de salto incondicional, cuando realmente son necesarias, se limitan específicamente a transferir el control dentro del mismo refinamiento. Las transferencias de control fuera del refinamiento se efectúan siguiendo estrictamente las estructuras definidas por los refinamientos.

En otras palabras, un programa escrito siguiendo esta metodología, contiene una primera parte formada únicamente por un gran "comentario estructurado" donde se especifican todas las relaciones entre los refinamientos. Luego en la segunda parte, se explicitan las acciones comentando solo los límites de cada una.

De este modo la documentación del programa se encuentra concentrada en un único lugar, al principio, lo que per-

mite tratarla como archivo, independiente del resto del programa. Sin lugar a dudas que un programa escrito en SOL sin la documentación respectiva, sería suficiente para producir un colapso entre los estructuralistas, ya que al nivel del lenguaje no es necesario más que concatenación y salto condicional como estructuras básicas.

Sin embargo, es precisamente esta documentación de los refinamientos la que nos da mayor riqueza de información; más aún, podríamos prescindir de las acciones y ser capaces de reconstruir el programa; la inversa, en cambio, sería impracticable la mayoría de las veces.

Con respecto a la enseñanza de programación, la metodología SOL ha tenido una influencia importante en la propia metodología de la enseñanza. La presentación de conceptos, en sí misma, encierra toda una estructura de refinamientos que disciplina el estudio tanto a educandos como educadores. No hemos realizado experimentos didácticos y estadísticos porque la población estudiantil y la disponibilidad de equipos no es lo suficientemente amplia como para asegurar la validez de alguna conclusión inferida estadísticamente. Si, en cambio podemos afirmar que en el diseño de programas por parte de docentes, la metodología SOL ha influído notoriamente en la calidad de los mismos.

EXPERIENCIA ATEC

En los párrafos siguientes se indica la experiencia obtenida durante el desarrollo de un sistema computerizado para un Municipio de la Provincia de Buenos Aires, donde si bien no se dispuso de un precompilador, se aplicó para el desarrollo de los programas la metodología SOL.

El proyecto comprendió, luego del análisis y dictado de normas para los procedimientos administrativos que regulan los sistemas, el desarrollo de 50 programas escritos en lenguaje COBOL, con más de 30000 líneas de código en total.

El objetivo principal consistió en encontrar para los programas estructuras de control simples que permitieran una rápida visualización del flujo de información y control. De este modo se lograría que la tarea de implementación, documentación y puesta a punto de los mismos se viera facilitada. Con ese criterio se adoptó la metodología SOL, aplicada en la forma que se describe a continuación:

Para la implementación de cada programa, a partir del 'nivel enunciado' se procedió a definir cada una de sus componentes, especificadas a través de sucesivos refinamientos, como se indica en la figura 3 .

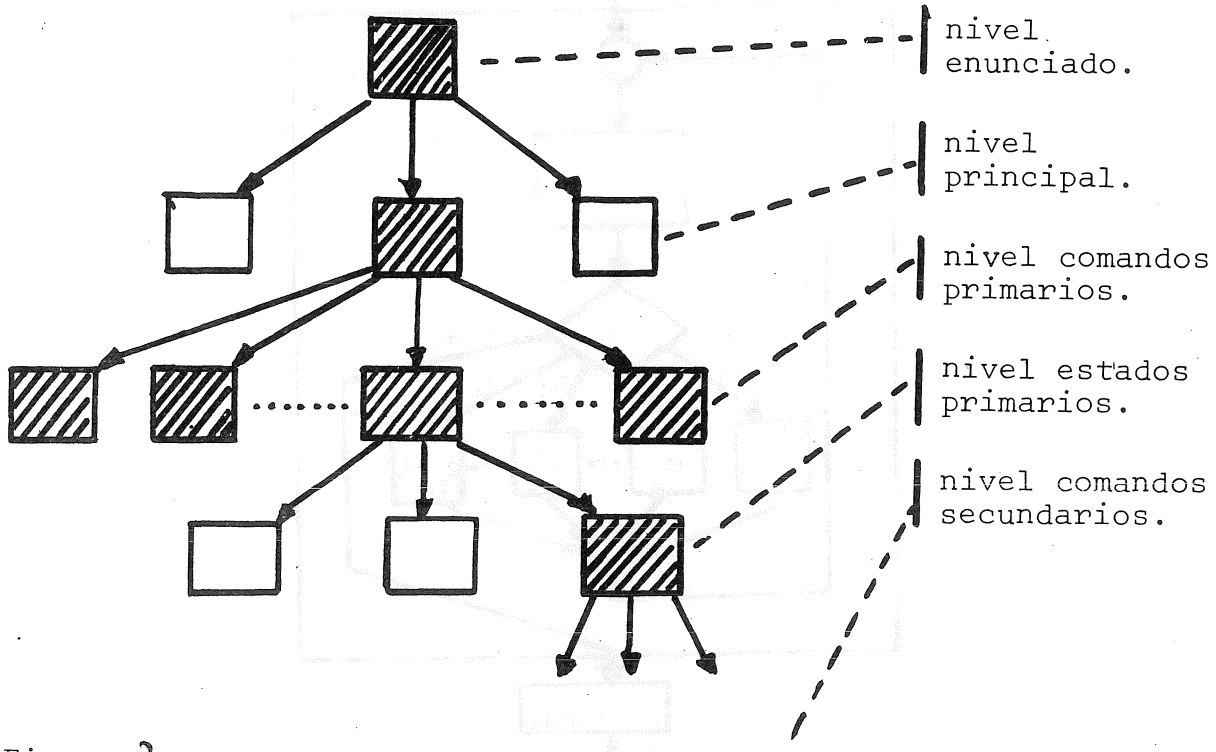


Figura 3

La primera etapa consiste en definir el 'nivel principal' del programa, mediante tres estados o módulos consecutivos, con la estructura indicada en la figura 4 y que en la sintaxis de SOL puede expresarse como se indica a continuación:

```
REF
  1  DO  inicializaciones, apertura de archivos y validaciones necesarias para la correcta ejecución del programa.

     THEN 2

  2  DO  desarrollo de todas las opciones que ejecuta el programa

     THEN 3

  3  DO  cerrar archivos, emitir estadísticas y terminar

     THEN STOP

END
```

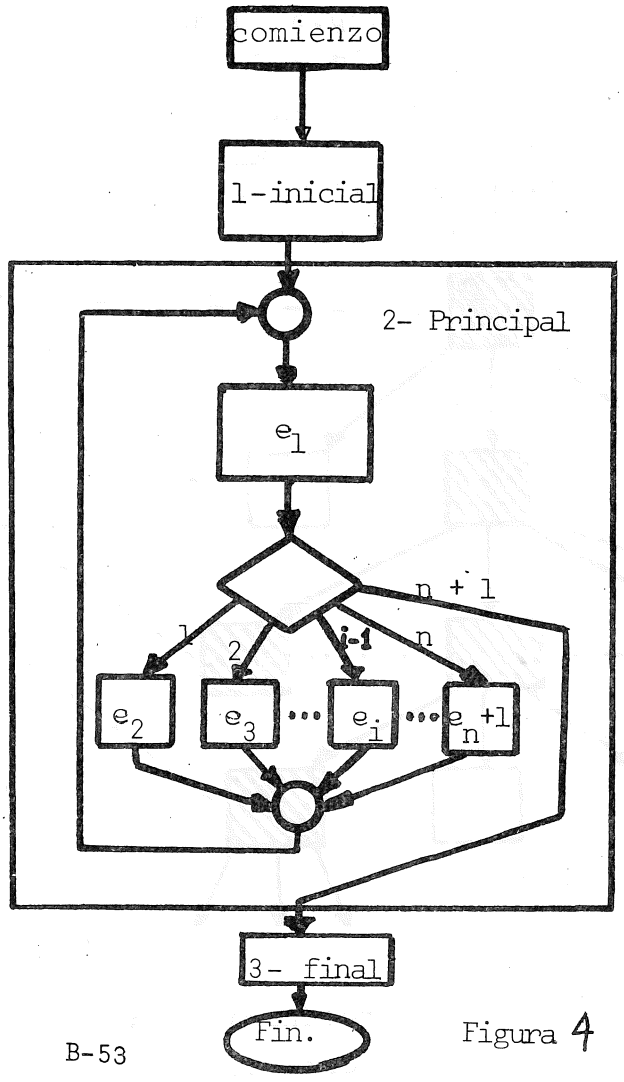


Figura 4

En la segunda etapa se refina el est. 2, módulo principal del programa, como un conjunto de estados, ejecutables de acuerdo a la entrada seleccionada por el operador o de acuerdo con condiciones internas del programa, dando origen al 'nivel comandos primarios'. En él, cada uno de los procesos que se llevan a cabo se los identifica con un estado denominado ek con k= 1,2,....,n,n+1. Se ejecuta inicialmente el estado e_1 y de acuerdo a la opción seleccionada por el operador o las condiciones internas detectadas por el programa, se ejecuta el estado e_i, con $2 \leq i \leq n+1$.

La entrada n+1 (que correspondería al est. e_{n+2}), queda reservada para la salida de este módulo, Según la sintaxis de SOL esta estructura puede definirse como se indica a continuación:

```
REF  2
  1   DO  seleccionar opción(1 a n para ejecutar estados 2
          a n+1 y n+1 para salir)
      THEN 2 OR 3 OR..... OR  i  OR...OR n+1 OR  EXIT1
  2   DO  ejecutar proceso correspondiente a opción 1
      THEN 1
  3   DO  proceso correspondiente a opción 2
      THEN 1
  .
  .
  .
  .
  i   DO  proceso correspondiente a la opción i-1
      THEN 1
  .
  .
  .
  .
  .
 n+1  DO  proceso correspondiente a opción n
      THEN 1
  END 2
```

En la tercera etapa cada uno de los estados componentes del 'nivel comandos primarios' se lo refina obteniéndose el

'nivel estados primarios'. Cada módulo es definido con la estructura que se indica en la figura5(el estado 3 se repite hasta que una condición de salida se produzca y retorna al nivel superior), y que en la sintaxis de SCL pueda expresarse, para i = 2,3,....,$n+1$, como:

```
REF  2.i

  1      DO  estado inicial
         THEN  2

  2      DO se sigue procesando? Afirmativo, estado 3. Negativo, estado 4.
         THEN 3 OR 4

  3      DO  realizar proceso correspondiente a este módulo
         THEN 2
  4      DO  finalizar y volver a refinamiento superior para
         nueva opción o terminar

         THEN EXIT1

END 2.i
```

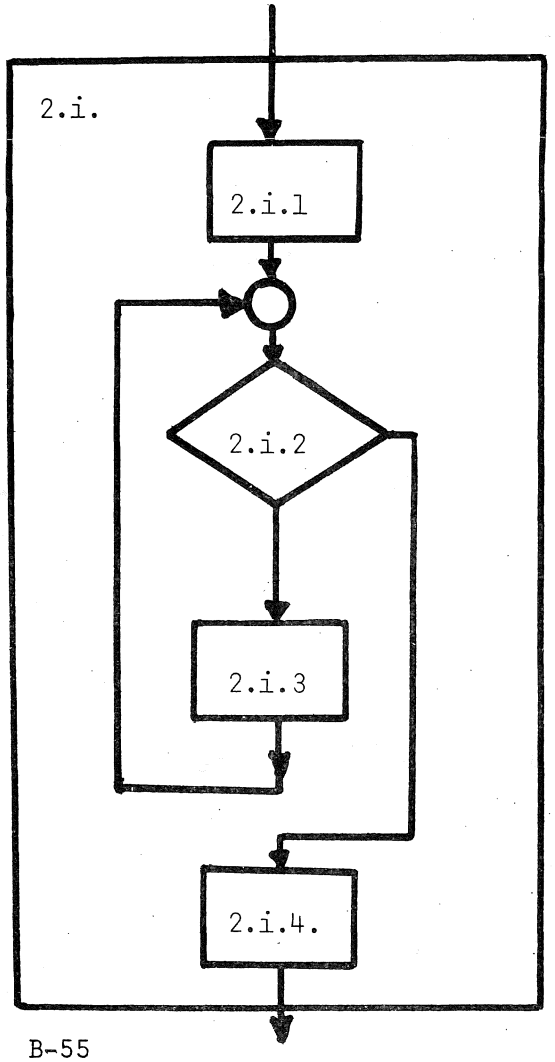


Figura 5

El próximo nivel refina al estado 2.i.3, que depende de cada caso en particular, En muchos casos, en este nivel hay estados implementados por rutinas que pueden ser utilizadas por más de un estado del mismo nivel, o estados similares de otros programas que pueden ser incluídos en el mismo en forma directa.

Al momento de la codificación, cada estado se concibió, dado que el lenguaje de programación era COBOL, como un módulo consistente en una SECCION invocado en el refinamiento superior por una sentencia PERFORM o un subprograma invocado mediante la sentencia CALL, desde el nivel superior.

Si bien se puede argumentar que una SECCION de programa invocada por un PERFORM no cumple todas las condiciones que exige la definición módulo independiente dado que no es capaz de compilarse independientemente, que puede interactuar libremente con otros sectores del programa y que no necesariamente tiene una definición de datos independientes, en el desarrollo que se describe, se prohibió que secciones o estados independientes interactuaran entre sí en otra forma que no fuera la especificada en los refinamientos. A cada SECCION se le asignó un grupo de datos de características locales, además de los globales pertenecientes al programa.

Otra norma adoptada fue la de prohibir que los GO TO's utilizados dentro de una SECCION se refirieran a marcas o rótulos externos a ella, con lo cual se logró independencia entre secciones.

Como corolario de esta experiencia se indica que en la medida que se respetaron las técnicas descriptas en el desarrollo de los distintos subsistémas, además del alto nivel de productividad de los programadores, se logró que la tarea deadaptación y mantenimiento de los programas no resultara una carga importante de trabajo. La estructura modular permitió que las modificaciones introducidas solo involucraran a un módulo en particular, sin extenderse hacia otros sectores, con la consiguiente generación de código confuso y poco claro. Por otra parte, dada la estructura común para casi todos los programas también lo fue la operación, redundando en una pronta adaptación del personal para el manejo de los sistemas.

EXTENSIONES PROPUESTAS

Actualmente parte del grupo de trabajo está implementando un sistema de programación interactiva utilizando la técnica SOL.

La manipulación de un programa parcialmente desarrollado, permitirá bajo ciertas condiciones, la posibilidad de ejecución simbólica de los refinamientos y la simulación de acciones aún no escritas.

Esta particularidad es atractiva, por cuanto el programador interactivo tendrá la posibilidad de ejecutar programas parcialmente desarrollados.

Una línea de trabajo propuesta, constituye la aplicación inversa de la metodología SOL a programas ya escritos. Este aspecto, denominado decompilación, permitiría rescatar en forma de refinamientos la estructura de control de un programa.

La aplicación de SOL al desarrollo de programas para Microprocesadores es parte también del objetivo de investigación.

En este sentido la relación entre programa y autómata se va estrechando para confundirse definitivamente en su implementación práctica.

CONCLUSIONES

Las experiencias logradas mediante la aplicación de la metodología SOL en diseño, desarrollo y mantenimiento de programas, permiten inferir mayor alcance tanto en el medio académico como profesional.

Las líneas de trabajo originadas por esta metodología suponen una intensificación del uso de la propia computadora para el desarrollo de programas. Esta programación interactiva se verá favorecida por la enorme proliferación de procesadores que se estima para la presente década.

Por otra parte el uso de metodologías como la propuesta sirven de marco a un medio unificado de programación, donde los programadores expresen sus ideas por medio de la estructura de refinamientos, independientemente del lenguaje a utilizar.

Este trabajo resume el estudio en una metodología original desarrollada y enriquecida durante los últimos tres años.

REFERENCIAS

1 CODAGNONE, J.A. "Implementación de un Precompilador SOL- BASIC". Informe Proyecto Final Dto. de Ingeniería - U.N.S. 1976.-

2 FONTAO,,R.O., ARDENGHI, J.R. y ARROYO, E.H.: "Sobre una Metodología de Programación". V Panel Computación y Expodata. Valparaíso, Chile (1978).-

APENDICE

LENGUAJE SOL

El lenjuaje SOL está orientado a mostrar explícitamente como se concibe la estructura de un programa mediante refinamientos similares a autómatas finitos.

El lenguaje mismo se diseñó para ser precompilado a un lenguaje práctico disponible. Las características sintácticas de SOL lo hacen especialmente adecuado para ser precompilado en un solo paso.

A continuación se dan resumidas las reglas sintácticas (no dependientes del contexto) que definen al lenguaje.

SINTAXIS

Se dan a continuación las producciones BNF para describir la sintaxis de SOL.

<programa> ::= <estructura de control> <descripción de acciones>

<estructura de control> ::= <refinamiento> FIN | <refinamiento> <estructura de control>

<refinamiento> ::= REF <identificador> <definición de AF> END <identificador>

<identificador> ::= λ | <identificador propio>

<identificador propio> ::= <entero positivo> | <entero positivo> • <identificador propio>

<entero positivo> ::= entero positivo sin signo

<definición de AF> ::= <descripción de estado> | <descripción de estado> <definición de AF>

<descripción de estado> ::= <entero positivo> DO <descripción - tarea elemental> THEN <lista de próximos estados>

<descripción-tarea elemental> ::= descripción en el lenguaje natural L de la tarea que se realiza en ese estado

<lista de próximos estados> ::= <identificador próximo estado> | <identificador próximo estado> OR <identificador próximo estado>

<identificador próximo estado>::=<entero positivo>|EXIT <entero positivo>|STOP

<descripción de acciones>::=<acción> FIN|<acción><descripción de acciones>

<acción>::=ACT <identificador><salida de estado><condición del estado>

<salida del estado>::=programa en el LPD

<condición del estado>::=NEXT|NEXT (<condición lógica>)

<condición lógica>::=condición lógica permitida en el LPD

SEMANTICA DE SOL

En lenguaje SOL, la estructura de un programa se define separadamente del resto de las acciones.

La estructura de control se compone,a su vez, de una sucesión de refinamientos (pasos 1 a n) concebidos como AFs. Los símbolos REF y END se usan para delimitar la definición de un refinamiento. El identificador se forma concatenando por medio de un punto (.) el identificador y el número del estado que será refinado. Así por ejemplo, si el estado 3 del refinamiento 3.2.4 debe ser refinado, entonces su identificador será 3.2.4.3. En particular, el primer refinamiento (paso 1) tendrá identificador nulo (Λ); es decir, el identificador del refinamiento de un estado del primer refinamiento será simplemente un entero (el número del estado que se refina).

Por consiguiente, la profundidad de un refinamiento se representa explícitamente en SOL por medio de una lista ordenada de enteros.

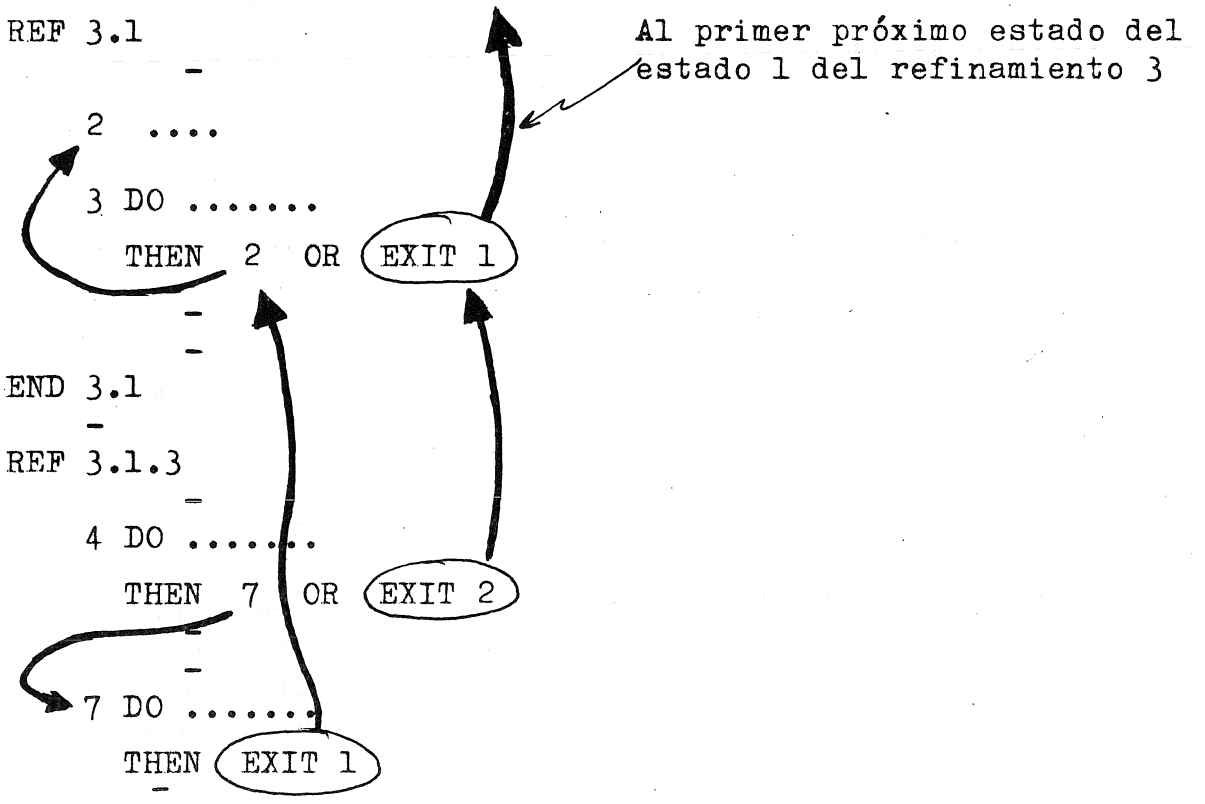
El estado inicial de un refinamiento será 1 y consecutivamente se numerarán los demás estados. Entre las palabras paréntesis DO y THEN se describirá el trabajo elemental a realizarse en cada estado. Esta descripción corresponderá a la acción (salida y condición) de la tabla de transiciones definida en la figura 1. El primer estado en la lista de próximos estados corresponderá a la entrada SI y el segundo (si existe) corresponderá a la entrada NO (figura 1).

El identificador del próximo estado puede ser de los siguientes tipos:

a) Entero positivo: identificará a un estado del mismo AF o refinamiento.

b) EXIT i: significa que el próximo estado será el indicado por el estado i-ésimo de la lista de próximos estados del refinamiento generador. (En el presente modelo i puede tomar solo dos valores 1 y 2).

c) STOP: significa que se ha llegado al estado final del AF.



Ejemplo de la semántica del próximo estado.

Con la palabra FIN termina la definición de los refinamientos. Estos forman un árbol cuyos nodos terminales representan los programas elementales que serán implementados a continuación.

Entre los símbolos ACT y NEXT se escribirán, las acciones correspondientes a las acciones del AF que modelará el programa. Sólo ACT lleva el identificador de la acción. Si NEXT no es seguida por una condición lógica entre paréntesis ello significa que hay un sólo próximo estado posible, de

otro modo el próximo estado estará indicado por el valor lógico de la condición: SI para el primero y NO para el segundo de los próximos estados.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

SISTEMA DE DESENVOLVIMENTO DE SOFTWARE DE MICROPROCESSADORES PARA APLICAÇÕES EM TEMPO REAL

J. Homero F. Cavalcanti
G. Singh Deep
R. Nazareno C. Alves
Depto. de Eng. Elétrica
UFPb.

1. OBJETIVO DO TRABALHO

Este trabalho tem o objetivo de mostrar um sistema de desenvolvimento de software de microprocessadores para aplicações em tempo real (SOFTR). O SOFTR permitirá o desenvolvimento de um sistema microprocessador, a partir de uma memória EPROM, que poderá ser usado em aquisição de dados, controle de processos industriais, etc. O SOFTR permite a interação usuário microprocessador via terminal vídeo/teclado. Além disto, as subrotinas do SOFTR poderão ser usados como suporte pelo sistema a ser desenvolvido. É apresentado o algorítmo, as subrotinas e a implementação do SOFTR para o M6800 da MOTOROLA.

2. INTRODUÇÃO

Atualmente, estudos estão sendo feitos para a utilização de microprocessadores em sistemas de aquisição de dados e controle de processos industriais [5]. Aplicações em tempo real geralmente necessitam de software capaz de responder a requisições de eventos externos. Estas requisições podem ser síncronas ou assíncronas [5], necessitando de software de controle especial baseados em processamento em lista [7], acionamento por interrupção [8], etc.

Aplicações de microprocessadores geralmente necessitam do desenvolvimento e implementação de um microcomputador. Os fabricantes de microprocessadores fornecem sistemas de desenvolvimento de software de aplicações tais como, o EXORCISER da MOTOROLA [2], o INTELLEC da INTEL [1], etc. Devido as dificuldades na obten

ção destes sistemas, algumas vezes é necessário o uso de "KIT'S" de desenvolvimento fornecidos pelos fabricantes (MEK-MOTOROLA [3], SDK-INTEL [1] etc). Estes KIT's, vêm acompanhados de um firmware (memória ROM) para o desenvolvimento de software do sistema desejado.

Apresentaremos um sistema de desenvolvimento de software de microprocessadores para aplicações em tempo real. O SOFTR foi desenvolvido para o M6800 da MOTOROLA baseado no programa de controle MIKBUG do MEK (MOTOROLA Design Evaluation KIT [2]) da MOTOROLA. O SOFTR foi desenvolvido com os seguintes objetivos:

a) facilitar o desenvolvimento de microcomputadores para aplicações em tempo real;

b) facilitar o manuseamento de interrupções pelo microcomputador;

c) permitir o desenvolvimento de sistemas de controle com multitasking [4,5,6] ;

d) permitir a interação entre usuário e o microcomputador;

e) permitir o uso da TTY (ou terminal vídeo/teclado) pelo usuário;

f) permitir o uso das subrotinas do SOFTR pelos programas de aplicação do usuário.

3. DESCRIÇÃO DO SOFTR

O SOFTR foi desenvolvido para aplicações em tempo real usando microprocessadores. A sua implementação foi feita no microcomputador EXORCISER da MOTOROLA [9] .

A figura 1 mostra a utilização do SOFTR num microcomputador. Neste esquema o microcomputador é composto de um bloco de memória RAM, uma EPROM (de 1k byte) para armazenamento do SOFTR, uma PIA (Peripheral Interface Adapter) para interface com periféricos (processos externos), e uma ACIA (Asynchronous communications Interface Adapter) para interface com a TTY ou terminal vídeo/teclado. A figura 2 mostra um possível mapa de memória para o microcomputador da figura 1. Neste mapa, o SOFTR (EPROM) é colocado do endereço FBØØ a FFFF (hexadecimal). A sua memória para armazenamento temporário de dados (RAM), de 128 bytes, é armazenada a partir do endereço FBØØ. A ACIA tem endereços FCF4 e FCF5.

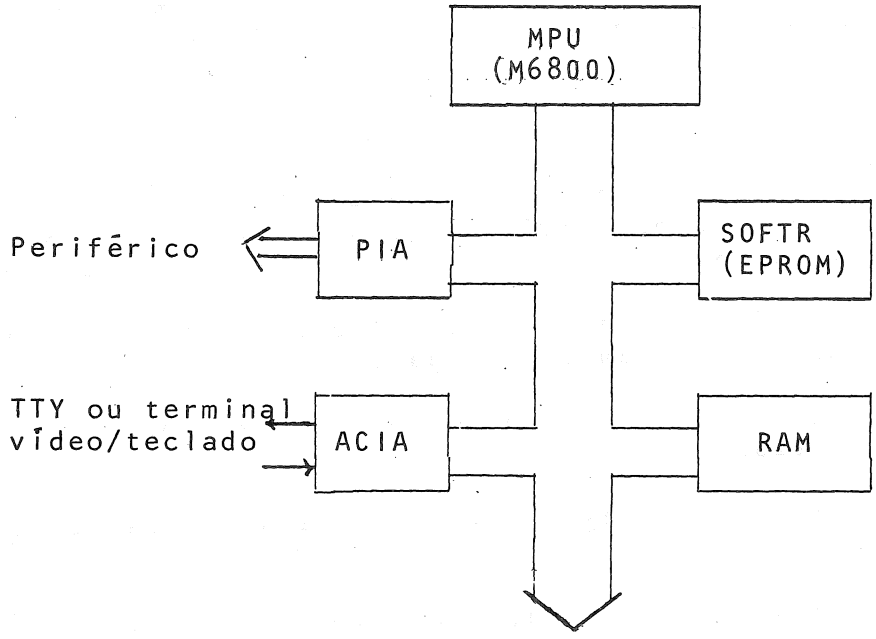


Figura 1

Esquema de um microcomputador usando o SOFTR.

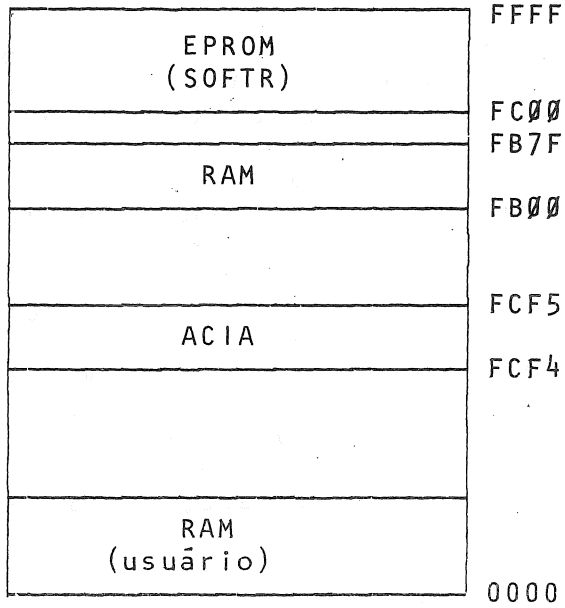


Figura 2

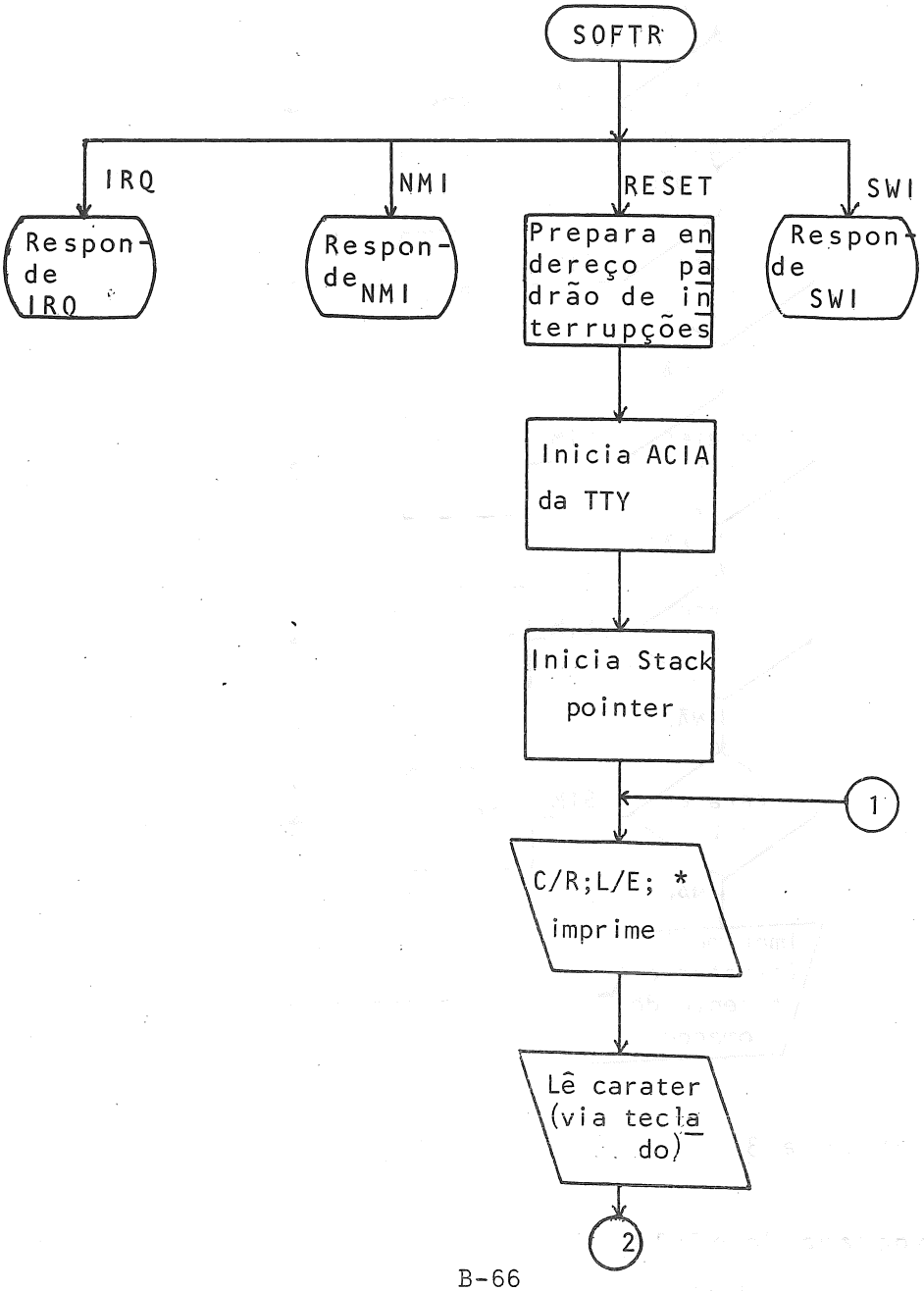
Mapa de memória do SOFTR

4. USO INTERATIVO DO SOFTR

A interação com o usuário é feita através de comandos como mostrados no fluxograma da figura 3.

Após o RESET, o controle é transferido ao SOFTR, um asterisco é impresso no início da próxima linha (TTY ou vídeo/teclado), e o SOFTR ficará esperando que o usuário tecle um dos comandos permitidos. Os comandos do SOFTR baseiam-se nos comandos do MIKBUG [3] da MOTOROLA.

Durante o funcionamento normal do sistema, o SOFTR poderá responder aos quatro tipo de interrupções do MOTOROLA [3]. O usuário tem total controle sobre estas interrupções.



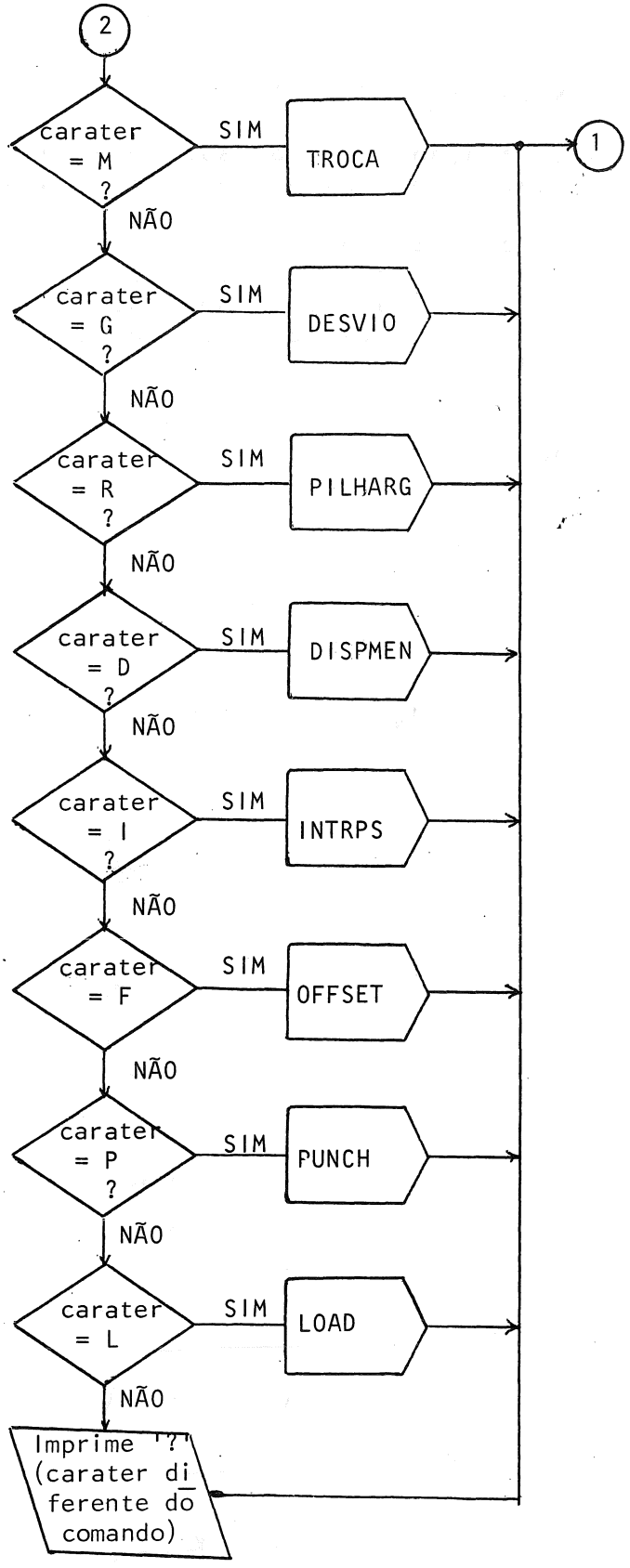


Figura 3

Fluxograma do SOFTR.

5. COMANDOS DO SOFTR

Os comandos são usados para a comunicação interativa entre o usuário e o SOFTR (ver figura 3).

5.1 - Examinar/Modificar conteúdo da Memória (comando M - subrotina TROCA)

Após o SOFTR imprimir (na TTY) um asterisco, o usuário tecla M. O SOFTR fica esperando (entrada via teclado) que o usuário tecle os 4 (quatro) números hexadecimais correspondente ao endereço que se quer examinar/modificar o conteúdo. O exemplo abaixo mostra a utilização deste comando.

```
*M Ø1ØØ
*Ø1ØØ 55 14
*Ø1Ø1 35 .
*
```

5.2 - Desvio para Programa do Usuário (comando G - subrotina DESVIO)

Após o SOFTR imprimir um asterisco o usuário tecla G seguido do endereço do programa a ser executado. No exemplo abaixo, o usuário deseja executar um programa a partir da posição Ø1ØØ.

```
*G  Ø1ØØ
```

5.3 - Mostra conteúdo dos Registradores (comando R - subrotina PILHAR)

Após o SOFTR imprimir um asterisco, o usuário pode verificar o conteúdo dos registradores do M6800 teclando R. O conteúdo dos registradores são mostrados na seguinte ordem: Registrador de Código de Condição (C); Acumulador B (B); Acumulador A (A); Registrador de Indexação (X); Contador do Programa (P); e Apontador da Pilha (S). O exemplo abaixo ilustra o uso deste comando.

```
*R  C-E5  B-D8  A-73  X-2345  P-1ØØØ  S-56ØØ
*
```

5.4 - DISPLAY (imprime) conteúdo de Locações de Memória (comando D - subrotina DISMEM)

Este comando permite a impressão (DISPLAY) do conteúdo de locações de memória indicadas pelo usário.

Após o SOFTR imprimir um asterisco, o usuário deverá teclar D, seguido do endereço inicial de memória e do endereço final de memória. O exemplo a seguir mostra o comando D para imprimir o conteúdo das posições de memória de Ø1ØØ a 010F (HEX).

```
*D0100 010F
*0100 8E018E0132FE0136C603960AA1022705095A ..2..6F...!./..Z
*
```

5.5 - Prepara Vetor de Interrupção (comando I - subrotina INTRPS)

Este comando permite a verificação e modificação do vetor interrupção (IRQ-SWI-NHI) do M6800. O SOFTR prepara endereços padrões para responder estas interrupções. O usuário, com este comando, pode definir um novo vetor de interrupção de acordo com as necessidades do seu sistema.

Após o SOFTR imprimir um asterisco, o usuário deverá teclar um I. O SOFTR imprimirá, na mesma linha, uma indicação da ordem das interrupções, e na próxima linha será impresso o endereço apontador da IRQ seguido do seu conteúdo (endereço da subrotina de interrupção atual). Se o usuário quiser modificar o endereço da subrotina de interrupção deve teclar um espaço seguido de quatro dígitos hexadecimais que representarão o novo endereço. Se o usuário teclar qualquer outro caráter, será apresentado o apontador da próxima interrupção, até serem apresentadas as três possíveis. No exemplo seguinte é mostrado uma possível utilização do comando I.

```
*I-IRQ-SWI-NMI
*FFF8 1000 0100
*FFFA 105F.
*FFFC 1100.
*
```

5.6 - Calcula Deslocamento para Desvios (comandos F - subrotina OFFSET)

O deslocamento é encontrado durante a montagem de programas usando-se código objeto. A montagem dos programas deve ser feita usando-se o comando M. Durante a montagem (ou verificação) do programa, quando for encontrado a locação do deslocamento, o usuário deverá retornar o controle ao SOFTR (teclando um caracter diferente de hexadecimal no campo do conteúdo da locação). Após o SOFTR imprimir um asterisco, o usuário deverá teclar um F seguido do endereço de desvio desejado. O exemplo abaixo mostra a determinação do deslocamento na montagem de um programa.

```
*M0102
*0101 FF .
*F 0120 1D
*0101 FF 1D
*0103 C5 .
*
```

5.7 - Perfura fita de Papel (comando P - subrotina PUNCH)

O comando P é usado para perfurar fitas de papel na TTY. Após o SOFTR imprimir um asterisco, o usuário deverá teclar um P seguido do endereço inicial e do endereço final das locações de memória dos dados a serem perfurados. Serão perfurados em grupos de 16 bytes de acordo com o seguinte protocolo.

```
S1 Ø6 Ø11B Ø1 33 39 7Ø
 1  2   3     4     5

S9 Ø3 ØØØØ FC
 6  2   3   5
```

Onde: 1 - cabeçalho do registro de dados
2 - byte de contagem
3 - campo do endereço
4 - conjunto dos dados
5 - checksum (complemento de um dos dados perfurados)
6 - cabeçalho do registro final.

A pefuração, numa fita, de dados armazenados nas locações de Ø1ØØ a Ø1ØF, é feita da seguinte forma:

```
*P Ø1ØØ Ø1ØF
S113Ø1ØØØ1Ø2Ø3Ø4Ø5Ø6Ø7Ø8Ø9ØØØ1Ø2Ø3Ø4Ø5Ø6A9
S9Ø3ØØØØFC
*
```

5.8 - Carrega Memória com Dados Perfurados em Fita de Papel (comando L - subrotina LOAD)

Após o SOFTR imprimir um asterisco, o usuário deverá colocar a fita de papel na leitora e teclar um L. Serão lidos registros de dados até se encontrar um registro final. Havendo detecção de erro no checksum, será impresso uma interrogação (?). Se o usuário desejar uma nova leitura do registro de dados, deverá reposicionar a fita e teclar um R. Caso contrário, deverá teclar qualquer caráter, exceto um R, para o controle voltar ao SOFTR. O exemplo a seguir ilustra o uso do comando L.

```
*L
S1Ø6Ø11BØ1323970?R
S1Ø6Ø11BØ1333970
S9
*
```

6. SUBROTINAS DO SOFTR

A tabela 1 mostra as principais subrotinas que podem ser utilizadas como suporte pelos usuários do SOFTR. A tabela é dividida em três campos: nome da subrotina (usando na codificação ASSEMBLY); endereço absoluto (referente a codificação ASSEMBLY

do código objeto do Apêndice A), função da subrotina.

NOME	ENDEREÇO	FUNÇÃO
SAICRT	113A	Envia à TTY o carater ASCII contido no ACCA.
ENTRAC	1125	Aceita um carater do teclado e o armazena no ACCA.
ENTHEX	10EB	Aceita um carater do teclado, transforma-o em hexadecimal e o armazena no ACCA.
LEBYTE	10D9	Aceita um byte de entrada e o armazena no ACCA.
FAZEND	10CB	Aceita dos bytes de entrada e os armazena no registrador X.
SAI2HX	110C	Saída de dois caracteres hexadecimais (apontados pelo registrador indecador-RX).
SAISPC	1107	Saída de um espaço.
SAI2HS	1105	Saída de dois caracteres hexadecimais mais espaço.
SAI4HS	1103	Saída de quatro caracteres hexadecimais mais espaço.
IMPD1	10C4	Saída de um conjunto de caracteres ASCII apontados pelo RX. O último carater deve ser 04.
RDOFF	1360	Desliga a leitora de fita de papel.

7. CONCLUSÃO

Neste trabalho apresentamos o projeto e desenvolvimento de uma EPROM dirigida para aplicações em tempo real. O SOFTR está sendo aplicado num sistema multimicroprocessadores atualmente em desenvolvimento na nossa Universidade.

Devido ao limitado número de páginas exigidas para apresentação, optamos por apresentar o código objeto (ver apêndice A) do SOFTR em vez da sua programação ASSEMBLY.

APÊNDICE A

O SOFTR foi testado no microcomputador EXORciser da MOTOROLA. Provisoriamente, foi armazenado da locação 1000 a 13FF (HEX) da memória do EXORciser. O programa tem início na locação 103F (HEX).

8. BIBLIOGRAFIA

1. INTEL, - "MCS-86 Product Descriptions", Fev. 1979.

2. MOTOROLA, - "M6800 Microprocessor Applications Manual", 1978.

3. MOTOROLA, - "MCM 6830L7 MIKBUG/MINIBUG ROM", 1976.

4. GONZALEZ, Mario J. - "Deterministic Processor Scheduling", ACM - Computing Surveys, Vol. 9, nº 3, set. 1977.

5. PARRISH, Edward A. Parrish; HUANG, Victor K.L. - "A Scheduler for Real-Time Task Control in Microcomputers", IEEE--IECI-25, nº 1, fev. 1978.

6. FULLER, Samuel H., JONH K, Ourster Hout, - "Multi-Microprocessors: And Overview and Working Example", Proceedings of the IEEE, vol. 66, nº 2, fev. 1978.

7. BAKER, Henry G. - "List Processing in Real Time on a Serial Computer", Communications of the ACM, vol. 21, nº 4, abril 1978.

8. ZELKOWITZ, Marvin, - "Interrup Driven Programming" Communications of the ACM, vol. 14, nº 6, junho 1976.

9. MOTOROLA, "EXORciser User's Guide", 1976.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

A ORGANIZAÇÃO DA PROGRAMAÇÃO

José Didyk Júnior
CELEPAR
Rua Mateus Leme, 1561 - Curitiba - PR - Brasil

I N T R O D U Ç Ã O

Em algumas das nossas empresas de processamento de dados, o desenvolvimento da programação tem sido relegado a um plano secundário nas preocupações relativas à qualidade do serviço prestado pelo C.P.D. . Entretanto, se analisarmos com alguma profundidade o impacto dos resultados obtidos no serviço de programação, veremos que os custos e a qualidade final do trabalho podem ser comprometidos em função de programas mal desenvolvidos.

Por outro lado, a utilização de técnicas modernas de programação, com o elemento humano bem treinado, concorre para a redução dos custos de desenvolvimento e manutenção de sistemas, além de oferecer um acréscimo significativo à segurança e confiabilidade dos sistemas.

Geralmente o trabalho do programador é visto simplesmente como uma decorrência do trabalho do analista, e por este fato não é suficientemente valorizado como uma atividade que pode contribuir significativamente para o sucesso dos sistemas em desenvolvimento.

Além destes aspectos, a carreira do programador parece, muitas vezes, terminar onde inicia a do analista, em termos de satisfação profissional, status e remuneração. Em função destes aspectos, o programador com dois ou mais anos de experiência, começa a manifestar interesse em tornar-se analista, o que vem provocar uma constante renovação dos quadros de programadores, acarretando uma falta constante de elementos especializados nesta área.

A partir da constatação destes fatos, nos propusemos a estabelecer condições para o desenvolvimento das potencialidades dos programadores em sua própria área de atuação, tornando-a mais valorizada perante as outras atividades do desenvolvimento de sistemas, e fazendo com que os programadores que realmente se interessem pela programação possam encontrar nessa carreira a oportunidade de crescimento técnico e realização profissional que desejam.

Este trabalho tem o escopo de relatar as conclusões obtidas a partir dos estudos e experiências realizados, bem como expor os resultados alcançados, procurando oferecer aos interessados no assunto, subsídios para o aperfeiçoamento dos métodos de trabalho utilizados.

I - CARACTERÍSTICAS DA ORGANIZAÇÃO

Podemos aceitar como provado que sem a existência de uma metodologia de trabalho adequada, dificilmente um determinado modelo de organização poderá apresentar bons resultados. Entretanto, neste trabalho, vamos nos ater aos aspectos organizacionais a serem considerados, visto que os pontos da metodologia, como linguagem adotada, padrões de desenvolvimento e manutenção , etc..., são considerados à parte como um pré-requisito para a utilização correta dos recursos existentes.

Antes de iniciar a definição de um modelo de organização para a programação, procuramos obter um levantamento das condições que deveriam ser levadas em consideração no estudo e planejamento da organização.

01. SEPARAÇÃO DA ANÁLISE

Uma das condições para o perfeito desenvolvimento da programação, seria a separação física da análise com a criação de um departamento ou equipe composta exclusivamente por programadores. Desta forma podemos orientar e conjugar os esforços dos programadores no sentido de aprimoramento das técnicas utilizadas. Além disso, fica facilitado o intercâmbio de idéias entre os programadores que podem assim aprender muito mais rapidamente, utilizando as experiências uns dos outros.

Outro aspecto deste arranjo, relaciona-se com as ligações da programação com a análise. Se os programadores estiverem reunidos sob uma chefia independente da análise, então eles terão oportunidades de competir com esta por uma imagem melhor dentro da empresa, de discutir prazos de programação, e de criticar os documentos relativos à definição de programas. Temos ainda uma facilidade maior para o controle das atividades do programador e do uso do seu tempo de trabalho, em virtude de maior precisão no acompanhamento das tarefas individuais e de grupos.

02. INCENTIVO À PESQUISA E EXPERIMENTAÇÃO

É um dos deveres da chefia de programação, incentivar o uso da pesquisa como ferramenta útil no desenvolvimento técnico dos programadores, e tam

bém como política de incorporação de novas técnicas. Através da pesquisa, que pode ser efetuada em diversos níveis, podemos obter grandes resultados para a melhoria dos recursos utilizados normalmente na programação: pelo conhecimento mais profundo dos softwares disponíveis na instalação, torna-se possível a recomendação do uso de recursos específicos, e a orientação de restrição ao uso de outros recursos que não sejam suficientemente otimizados, quando podemos substituí-los por outros. Aparecem também oportunidades de desenvolvimento de softwares específicos para o uso da empresa, que podem representar um grande impacto em termos de economia.

Neste processo, tem papel preponderante a atuação da chefia de programação, que precisará orientar os programadores no desenvolvimento do seu esforço de pesquisa, fazendo com que os resultados sejam divulgados, e que os assuntos pesquisados sejam de interesse para a empresa.

03. PERSPECTIVAS DE CRESCIMENTO PROFISSIONAL

Muitos programadores que hoje são analistas de sistemas teriam talvez permanecido na programação, caso tivessem uma melhor perspectiva de crescimento profissional nesta área. Entretanto, normalmente a carreira de analista aparece como uma opção lógica e imediata de progresso para os programadores, em função das limitações geralmente impostas pelas empresas. Isto acarreta que pessoas que normalmente se identificam com a programação procurem seguir carreiras nas quais não irão se adaptar perfeitamente, mas onde poderão obter uma remuneração mais compensadora.

Podemos também encontrar com facilidade lugares onde a programação não evolui tecnicamente, o que leva o programador a procurar novas atividades que possam proporcionar uma evolução profissional mais compensadora. Deduzimos daí a necessidade de oferecermos não apenas uma possibilidade de remuneração mais elevada, mas também um incentivo ao desenvolvimento de novas técnicas e oportunidades de atualização.

Para que possamos proporcionar ao programador perspectivas de desenvolvimento e atualização profissional, precisamos dispor de um plano de treinamento adequado às necessidades da programação e aos recursos da empresa.

Grande parte do treinamento de programação pode ser levado a efeito a partir do aproveitamento de recursos já existentes, sem a necessidade de envolvimento de entidades estranhas à empresa. Os programadores mais experientes podem ser utilizados como instrutores para o pessoal de programação, a partir do estudo e pesquisa de assuntos de interesse geral e posterior apresentação dos resultados em uma palestra ou curso. Este sistema oferece como vantagem uma grande objetividade no treinamento, visto que o mesmo é desenvolvido por uma pessoa que possui vivência no ambiente onde os objetivos devem ser alcançados.

04. APROVEITAMENTO DOS RECURSOS HUMANOS

A distribuição do trabalho na programação é geralmente bastante problemática, pois deve levar em consideração os prazos, a complexidade do pro

grama e a qualificação do programador. Na atividade de manutenção é normalmente mais produtiva a alocação de programadores que já conhecem o sistema ou programa a ser alterado, de forma a diminuir o tempo gasto na análise do problema, e obter ainda uma segurança maior neste trabalho.

O modelo de organização deve levar em consideração estes aspectos, de modo a fornecer alternativas de alocação de programadores para desenvolvimento de novos sistemas e para as atividades de manutenção prevista ou de caráter excepcional, procurando sempre obter distribuição homogênea para a carga de trabalho dos programadores.

05. AUMENTO DA ABRANGÊNCIA DA PROGRAMAÇÃO

No trabalho de projeto, desenvolvimento, implantação e manutenção de sistemas, existe um determinado número de tarefas que são normalmente divididas entre a análise e a programação. Porém, algumas tarefas normalmente executadas pelo analista poderiam ser desenvolvidas por programadores experientes, a partir de um pequeno período de treinamento. Entre elas estão a definição de programas, projeto de arquivos e lay-outs, acompanhamento e suporte à programação, teste de sistemas, e ainda, o suporte à operação na fase de implantação. Nos sistemas já implantados, a análise dos problemas ocorridos em produção pode ser efetuada em grande parte por um programador com um conhecimento detalhado do sistema e de seus programas.

Esta distribuição de atribuições entre análise e programação produz não apenas uma diminuição dos custos do desenvolvimento de sistemas, como ainda permite ao analista uma alocação de uma parcela maior de seu tempo para o cliente. Concorre também para tornar a programação mais segura em seus resultados, em virtude da obtenção de um conhecimento mais profundo e abrangente sobre o sistema.

A partir da adoção desta nova distribuição das tarefas no desenvolvimento de sistemas novos, a programação passa a produzir como resultado final de seu trabalho, sistemas já prontos para a implantação sem a necessidade de testes adicionais.

06. CARACTERÍSTICAS DA CHEFIA

Além da liderança pessoal, a chefia imediata dos programadores deve exercer uma liderança técnica bastante acentuada, de forma a fazer com que os programadores possam encontrar ali a orientação necessária especialmente na fase de formação e treinamento.

Outro papel importante da chefia refere-se à responsabilidade de despertar e orientar em seus subordinados o interesse pela pesquisa, proporcionando os recursos necessários para o desenvolvimento das mesmas, assumindo, sempre que possível, a participação na pesquisa como orientador.

07. ÊNFASE NO TRABALHO EM GRUPOS

A técnica de trabalhos em grupos oferece grandes vantagens para a programação, pois torna mais fácil a disseminação dos conhecimentos específi-

cos sobre a linguagem, técnicas e metodologia adotadas. Especialmente na formação e desenvolvimento dos programadores menos experientes, o trabalho em grupos com programadores experientes acelera o desenvolvimento técnico, facilitando o aprendizado, e a identificação e correção de falhas no processo de formação dos programadores.

Ainda podemos citar como vantagem dos trabalhos em grupos o maior entrosamento e conhecimento dos programadores sobre os sistemas e programas que estejam sob sua responsabilidade, facilitando a troca de informações e experiências sobre os mesmos. Isto impacta na identificação de falhas ou omissões existentes nos programas do sistema considerado como um todo, oferecendo como resultado imediato maior segurança no desenvolvimento dos programas.

II - A ESTRUTURA

Utilizando os itens relacionados no capítulo anterior como ponto de partida para o estudo e pesquisa de um modelo de organização para a programação, chegamos a um resultado que se baseia na existência de uma gerência e várias coordenações de equipes, cujas atribuições são descritas a seguir:

GERÊNCIA DE PROGRAMAÇÃO

Tem a seu cargo a participação na definição das diretrizes da empresa e a representação junto aos outros órgãos, dos interesses da programação nas áreas que afetam seus procedimentos. Em função desta participação a gerência deve determinar as diretrizes internas à programação, no sentido de melhor contribuir para atingir os objetivos estabelecidos pela empresa.

A gerência deve também promover o desenvolvimento dos métodos em uso pelo departamento, solicitando e orientando trabalhos de pesquisa sobre metodologias, software, etc. A partir dos resultados destes trabalhos ou de solicitação dos coordenadores de equipe, analisa, acompanha e promove as atividades de treinamento.

A distribuição de trabalhos para as coordenações é planejada a nível de sistema pela gerência, que participa da definição de prazos e estimativas de custos da programação, bem como do controle do desenvolvimento dos trabalhos.

COORDENAÇÃO DE PROGRAMAÇÃO

As coordenações devem ser formadas por um coordenador e até nove programadores, sendo três programadores nível "pleno" ou "senior", três nível "júnior" e três "trainee", podendo esta composição ser modificada dependendo das características do trabalho.

O coordenador deve possuir conhecimentos administrativos, além de formação em análise e programação, tendo sob sua responsabilidade a administração e supervisão dos trabalhos desenvolvidos na sua equipe. Suas principais atribuições são:

- Orientação e suporte técnico para os programadores no desenvolvimento e manutenção de sistemas;
- Distribuição da carga de trabalho e alocação de recursos na equipe;
- Planejamento e acompanhamento das atividades de grupo e individuais dos programadores;
- Acompanhamento da evolução técnica de cada programador, com vistas à detecção de neeessidades de treinamento ou reciclagem;
- Acompanhamento do uso e cumprimento das normas e padrões adotados pela empresa;
- Orientação e acompanhamento para o desenvolvimento de pesquisas na equipe;
- Verificação e aprovação da qualidade do trabalho desenvolvido pela equipe;
- Avaliação do desempenho individual dos programadores com vistas à indicações para promoções;
- Estabelecimento de prazos e estimativa de tempos para o desenvolvimento de sistemas, no que se refere às atividades de programação.

Dentro de cada coordenação de programação, os programadores são enquadrados em classes funcionais, de acordo com sua experiência e habilitação, a saber:

Programador trainee - sem experiência
Programador junior - mínimo de 1 ano de experiência
Programador pleno - mínimo de 2,5 anos de experiência
Programador senior - mínimo de 4 anos de experiência

Além do desenvolvimento e manutenção de programas, os programadores nível "senior" e "pleno" têm a seu cargo, no caso de sistemas em desenvolvimento, as atividades de elaboração do fluxo do sistema, definição de arquivos e programas, e execução de testes de sistema, necessitando formação adicional com vistas ao desempenho destas atividades. Já os programadores nível "junior" e "trainee" devem apenas desenvolver e prestar manutenção em programas, sendo que os "junior", à medida em que começam a apresentar maior desenvolvimento, devem ser iniciados nas funções atinentes ao cargo de "pleno".

O número de sistemas que podem ser mantidos em cada coordenação está em torno de 15, divididos em dois a três sistemas de grande porte (acima de 50 programas), oito de médio porte (entre 20 e 50 programas), e cinco sistemas de pequeno porte (até 20 programas), considerando-se o número médio de linhas de código por programa em torno de 1.500. Além deste trabalho de manutenção, cada coordenação pode desenvolver simultâneamente dois ou três sistemas novos, de porte médio.

III - O MODELO

Dentro deste modelo de organização da programação, cada coordenação tem a seu cargo o desenvolvimento e manutenção dos programas de diversos sistemas. Como o desenvolvimento e a manutenção de um sistema são atividades de características bastante diferentes, analisaremos as duas separadamente:

01. DESENVOLVIMENTO DE NOVOS SISTEMAS

Quando do desenvolvimento de um novo sistema, este deverá sempre ser efetuado por uma única coordenação, que terá a responsabilidade de entregar como resultado final de seu trabalho, o sistema já testado e pronto para implantação.

O primeiro passo no desenvolvimento de um sistema na programação seria a indicação, pela gerência, de qual coordenação terá disponibilidade para assumir a carga horária representada por este trabalho.

Após a decisão da gerência, o coordenador da equipe indicada seleciona um programador de nível "pleno" ou "senior", que será o responsável técnico pelo sistema enquanto o mesmo estiver na programação, e até que receba a aprovação final da análise.

Dependendo do tamanho e complexidade do sistema em questão, poderão ser alocados mais um ou dois programadores de alto nível para trabalhar em conjunto com o programador escolhido anteriormente, porém o responsável será sempre um só, estando os outros subordinados a ele em termos técnicos, no tocante ao desenvolvimento daquele sistema. Uma vez definido o programador responsável pelo sistema, e os programadores que deverão auxiliá-lo se for o caso, o trabalho é dividido em três fases distintas, que compreendem : definição do fluxo, arquivos e programas do sistema, desenvolvimento e testes dos programas, e finalmente teste do sistema e aprovação final.

01.01. DEFINIÇÃO DOS ARQUIVOS E PROGRAMAS

Durante o desenvolvimento desta fase do trabalho, serão envolvidos apenas o analista responsável pelo sistema e os programadores indicados até o momento.

A partir dos documentos de análise e das especificações para o sistema, o programador responsável e seus auxiliares, em conjunto com o analista, deverão produzir a definição do fluxo e dos arquivos do sistema.

Nesta etapa, além de se obter um resultado necessário para a continuidade dos trabalhos, os programadores ficam conhecendo o sistema e os dados que o mesmo irá manusear, o que é indispensável para que possam dar início ao próximo passo, que trata da definição dos programas.

Para a definição dos programas, são utilizados os documentos produzidos até o momento sobre o sistema, e ainda os conhecimentos adquiridos pelos programadores na atividade anterior.

Cabe neste ponto ressaltar uma vantagem adicional obtida com este método, que consiste na necessidade de uma documentação de análise sobre as etapas anteriores, o que muitas vezes é feito pelo analista apenas ao final do desenvolvimento do sistema, e que pode causar a perda de muitas informa-

ções úteis sobre o sistema.

Via de regra, dependendo do tamanho do sistema, as definições de programas são divididas de acordo com as rotinas do sistema (como consistência, compatibilidade, cálculos, relatórios, etc), de forma a obter-se resultados intermediários, que são submetidos à apreciação e aprovação do analista.

Além da definição dos programas do sistema, é produzido, para cada programa um fluxograma estruturado, a nível de funções. Cada função é considerada isoladamente na descrição de procedimentos, de modo a obter-se uma perfeita separação entre elas. O resultado deste trabalho é uma definição de programa que fornece um esquema da lógica que deverá ser utilizada na confecção do programa.

Mediante a utilização deste recurso, conseguimos uma redução quase total dos erros ocasionados por problemas de entendimento ou interpretação das definições de programas, e obtemos uma padronização da lógica dos programas de cada sistema, o que vem facilitar o trabalho de manutenção. Podemos acrescentar ainda que a análise de implementação de alterações nos programas fica bastante facilitada, em virtude da separação das funções existentes.

Após a conclusão da definição dos programas de cada rotina do sistema, estas são submetidas à apreciação do analista, que deverá verificar a correção das definições autorizando o início da confecção dos programas.

01.02. DESENVOLVIMENTO E TESTE DOS PROGRAMAS

A exemplo do que ocorre na etapa anterior, o desenvolvimento e teste dos programas é também executado levando-se em consideração as rotinas do sistema.

Nesta fase do trabalho são alocados mais programadores, (agora de nível "junior" e "trainee"), que deverão desenvolver e testar individualmente os programas já definidos.

Estes programadores irão trabalhar sob a supervisão do programador responsável pelo sistema, que deverá orientá-los nas dúvidas que vierem a ocorrer, e ainda irá supervisionar a execução dos testes de programa. Em virtude do conhecimento que o programador responsável tem sobre cada programa e sobre o sistema em geral, obtemos resultados altamente compensadores, tanto em termos de confiabilidade dos programas, como na economia dos recursos de máquina e pessoal necessários para a confecção dos programas.

Cada programa deve ser codificado de maneira a seguir rigidamente a estrutura planejada para ele na sua definição. Caso seja necessária uma alteração nesta estrutura, a mesma somente poderá ser efetuada com a concordância do programdor responsável pelo sistema, e mediante alteração da definição original. Os programas devem também receber a aprovação do programador responsável pelo sistema, que deve certificar-se de que o programa foi desenvolvido em acordo com as normas e padrões da empresa.

No caso de programas complexos, o programador responsável participa do detalhamento da lógica do programa e do planejamento dos testes, que devem ser também executados dentro de padrões pré-determinados.

01.03. TESTE DO SISTEMA E APROVAÇÃO FINAL

Após a conclusão das etapas anteriores, deve ser iniciado o teste integrado do sistema, que consiste na execução de todas as rotinas e programas, visando simular as condições normais de utilização do mesmo em produção.

A primeira atividade desta etapa é a obtenção dos arquivos de testes. Para tanto, são utilizados os documentos fonte do sistema, preenchidos de modo a obter arquivos que testem todas as possibilidades de ocorrência de dados para o sistema. Esta tarefa é executada pelo programador responsável, que pode ser auxiliado por outros programadores quando necessário.

Ao término da fase de testes, devem ter sido executados todos os comandos dos programas, e devemos ter todos os tipos de saídas do sistema. Isto feito, os resultados obtidos devem ser submetidos à apreciação do analista para a aprovação final, quando o sistema será considerado entregue pela programação. A partir deste ponto, apenas o programador responsável poderá ficar alocado ao sistema com a finalidade de acompanhar a etapa de implantação.

01.04. LIBERAÇÃO DOS PROGRAMADORES

Os programadores nível "junior" e "trainee" são liberados logo após a conclusão dos programas, com aprovação do programador responsável.

Contudo, de vez que o pessoal envolvido no desenvolvimento do sistema, em especial o programador responsável, adquire um know-how sobre o mesmo, procuramos obter um fruto a mais desta situação, no que se refere ao esquema de manutenção do sistema após a implantação do mesmo.

Os programadores que participaram da definição dos programas ficam alocados para manutenção do sistema e solução dos problemas que eventualmente venham a ocorrer em produção. Desta forma, temos normalmente duas a três pessoas que conhecem o sistema como um todo, e que podem prestar uma assitência rápida e eficiente para a solução de problemas imprevistos ou em caso de manutenção planejada.

A decisão de qual programador irá atender os casos de necessidade de manutenção fica a critério do coordenador de equipe, que decide baseado na disponibilidade de tempo de cada programador, e no seu envolvimento com outros sistema.

02. MANUTENÇÃO DE SISTEMAS

Uma boa parcela do tempo gasto pela programação durante os trabalhos de manutenção, refere-se à aquisição, pelo programador, de um know-how sobre o programa e o sistema, que lhe permita efetuar as alterações necessárias.

Entretanto, no caso de sistema desenvolvidos dentro deste modelo, teremos sempre dois ou três programadores que conhecerão os programas e o sistema, e para os quais o tempo gasto em estudo e aquisição de novos conhecimentos é bastante reduzido, senão nulo. Desta forma podemos dispor de um programador de alto nível que, se não tiver disponibilidade para executar a manu-

tenção solicitada, poderá orientar outros programadores no desempenho desta tarefa.

Isto vem conferir uma rapidez e segurança aos trabalhos de manutenção que dificilmente poderia ser conseguida de outra maneira, sem um grande dispêndio de tempo.

No caso de sistemas mais antigos, desenvolvidos segundo outros métodos, a cada vez que seja solicitada uma manutenção, deve ser alocado um grupo de programadores, geralmente dois ou três, que irão estudar e conhecer o sistema e seus programas para efetuar o serviço de manutenção.

Desta maneira serão formados grupos de programadores, que terão um conhecimento bastante profundo sobre um conjunto de sistemas, e estarão aptos a prestar rapidamente as manutenções necessárias.

Com o passar do tempo, os custos de manutenção destes sistemas tendem a diminuir em função da maior rapidez e segurança com que serão executados.

Os programadores responsáveis por um conjunto de sistemas podem ser alocados para outros serviços de desenvolvimento e manutenção, sempre que a carga de trabalho dos mesmos permita tal envolvimento. Obtemos assim uma boa distribuição de carga de trabalho dentro das equipes, o que proporciona um alto nível de produtividade.

IV - CONSIDERAÇÕES FINAIS

O modelo exposto neste trabalho tem se mostrado bastante superior aos anteriormente utilizados, tendo resultado em um aumento significativo na produtividade da programação, em especial no que se refere à segurança dos resultados obtidos ao final de cada trabalho. Além deste aspecto, detectamos outros fatores que nos pareceram relevantes, e que seguem relacionados abaixo:

01. MELHORIA DA DOCUMENTAÇÃO

Em função da necessidade de transmissão para a programação das informações resultantes do trabalho de análise, a documentação do sistema torna-se mais completa, pois neste aspecto o trabalho desenvolvido pelo programador responsável funciona como um ponto de verificação da documentação, o que não acontecia anteriormente com tal profundidade.

02. EVOLUÇÃO TÉCNICA DOS PROGRAMADORES

O envolvimento do programador em diversos programas e sistemas de características diferentes, com a supervisão de pessoal experiente, conduz a um aprendizado rápido e eficiente das técnicas mais avançadas de programação. A variedade das situações que se apresentam fazem com que o programador obtenha uma vivência intensa e diversificada, e tome contacto com soluções que se baseiam na experiência de outros programadores com visão mais ampla e precisa sobre os programas e sistemas em questão. O programador vai também adquirindo com esta prática e capacidade de analisar os problemas a nível de sistema, não se restringindo apenas aos programas.

Em virtude do desenvolvimento de trabalhos em grupo, os programadores têm a oportunidade de trocar conhecimentos uns com os outros, o que vem ainda possibilitar a cada indivíduo a conscientização de sua situação, em termos técnicos, perante os outros programadores.

03. LIBERAÇÃO DO TEMPO DA ANÁLISE

Em função dos prazos e das tarefas a serem executadas durante o desenvolvimento de novos sistemas, o contacto do analista com o cliente é às vezes prejudicado, afetando a obtenção de resultados para a empresa. Com a programação assumindo a responsabilidade do desenvolvimento de arquivos e definições de programas e ainda do teste de sistema, o analista pode dedicar uma parcela maior de seu tempo ao atendimento do cliente, fato que impacta na satisfação do mesmo, proporcionando uma afinamento melhor entre as espectativas e o relacionamento empresa/cliente.

No atendimento a problemas ocorridos em produção, grande parte pode ser resolvido pelo programador sem necessidade de envolvimento do analista, que será utilizado apenas quando estes problemas impactarem em alterações no sistema ou estiverem relacionados com as rotinas ou documentos do cliente.

04. SATISFAÇÃO PROFISSIONAL DO PROGRAMADOR

Em grande parte das empresas a programação é utilizada como fonte de obtenção de recursos humanos para outros departamentos como análise, suporte técnico, etc. Este arranjo é benéfico quando proporciona a abertura de novas perspectivas de evolução para o profissional de programação, que pode vir a identificar-se com outras atividades, encontrando aí a possibilidade de um entrosamento perfeito entre suas aspirações e a atividade que desempenha. Por outro lado, muitos programadores acabam tornando-se analistas não por força da adequação desta atividade às suas expectativas profissionais, mas buscando apenas uma remuneração ou um "status" mais elevados. Quando isto acontece, podemos estar patrocinando a criação de profissionais insatisfeitos e deslocados em seu ambiente de trabalho e nas atividades que desempenha. Mediante a adoção de uma política salarial mais ampla e de valorização das funções da programação este problema será eliminado, visto que estarão sendo eliminados os fatores que provocam o aparecimento do mesmo. Continuaremos a ter programadores que irão manifestar o desejo de tornarem-se analistas, mas agora isto irá ocorrer a partir da constatação pelo indivíduo de que esta atividade é mais adequada às suas características pessoais, proporcionando melhores condições para a obtenção da realização pessoal que ele almeja obter.

O aumento das atribuições do programador, representado pelo desenvolvimento de sistemas desde a fase de definição de arquivos até o teste do sistema, vem patrocinar a oportunidade do programador avaliar os resultados do seu trabalho como um produto acabado, o que promove a valorização individual ao permitir a visualização da importância do trabalho executado, para a realização dos objetivos propostos no desenvolvimento de sistemas.

05. CONCLUSÃO

Apesar de ter sido apresentado dentro de um contexto empresarial que comporta a existência de uma gerência de programação, este aspecto pode ser modificado de acordo com a estrutura das empresas que se disponham a adotar este modelo de organização. O número de coordenações existentes deve ser dimensionado a partir do volume de trabalho na programação, de maneira a acompanhar o crescimento da demanda de mão-de-obra necessária na programação.

Nos países com um tradição maior em processamento de dados, os programadores têm a seu cargo responsabilidades e tarefas mais amplas do que têm tido até o momento na maioria das empresas brasileiras. Certamente a disponibilidade de equipamentos e pessoal em nossas empresas não pode ser comparada com o equivalente em países mais desenvolvidos. Porém, a exploração das potencialidades dos indivíduos tem sido efetuada em um nível relativamente baixo, quando comparada com aquela efetivamente realizada nestes países. Este fato leva-nos a supor que podemos atingir níveis de produtividade mais altos, que proporcionem às empresas um retorno melhor sobre o custo do pessoal especializado, a despeito de não podermos oferecer sempre os recursos de máquina e software ideais.

Este trabalho representa o resultado de um esforço efetuado no sentido da obtenção de níveis de produtividade e satisfação pessoal mais altos que aqueles existentes anteriormente na programação, tendo até o momento apresentado um resultado altamente compensador em virtude dos progressos alcançados.

B I B L I O G R A F I A:

. Metodologia CELEPAR - MC-DMS - CELEPAR

. Manual da Organização - CELEPAR

. Manual de Normas e Procedimentos da Programação - CELEPAR

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

DESENVOLVIMENTO E TESTES DE PROGRAMAS ESTRUTURADOS

Eduardo Ferreira Eleotério
CELEPAR
Rua Mateus Leme, 1561 - Curitiba - PR - Brasil

I N T R O D U Ç Ã O

Em uma Empresa de Processamento de Dados, normalmente encontramos os mais variados métodos de se confeccionar um programa, no qual irá impactar diretamente em futuras manutenções dos mesmos.

Em estudos realizados ao longo do tempo nos mostrou que esta variação era obtida através de métodos próprios adquiridos pelos programadores, dificultando a alteração de um programa que tivesse sido feito por um outro programador.

Preocupados em unificar o desenvolvimento de programas, pesquisamos e chegamos a conclusão que seria necessário a aplicação de um método que utilizado por vários programadores, obtivessemos uma solução única. Dentro dos estudados, concluímos que seria adequado para nossas necessidades, os métodos de programação estruturados, no qual obtivemos pleno êxito em nossas aplicações. E este trabalho mostra como desenvolvemos e testamos programas de forma estruturada.

I - ESTUDO DA DEFINIÇÃO

Esta etapa dos procedimentos do programador tem como objetivo, propiciar ao mesmo tempo, um conhecimento detalhado e seguro dos dados a serem manuseados pelo programa, e dos procedimentos necessários para a transformação destes.

Para tanto, é necessário que a definição esteja completa, pois, é importante que se tenha todas as informações para o desenvolvimento do programa.

Ao estudar a definição, o programador deverá entender a lógica do programa, anotando à parte todas as dúvidas que surgirem, fazer um levantamento e esclarecer todos os pontos obscuros. Estes procedimentos devem ser feitos, tantas vezes quanto for necessário, até que tenha compreendido a de

finição por completo, evitando assim interpretações errôneas.

Como o programa será desenvolvido através de um método estruturado, o programador deverá ter uma preocupação adicional com os dados de entrada e saída do seu programa, sendo este procedimentos fundamental para a aplicação do método.

Como resultado final desta etapa, o programador terá o controle e conhecimento necessário do programa, facilitando sobremaneira a continuidade do desenvolvimento.

II - ESTRUTURAÇÃO LÓGICA

A finalidade da estruturação lógica, é obter solução lógica para o problema representado pelos dados a serem processados, e os procedimentos necessários para esta transformação. E, para obter esta solução, devemos utilizar um método estruturado, no qual nos solucione o problema sem que haja a preocupação a nível de detalhe.

Dentro de experiências realizadas em nossa empresa, chegamos a conclusão que o método de JACKSON supria bem nossas necessidades, pois consiste em estruturar um programa baseado nos dados de entrada e saída do programa, e mostrou também uma simbologia variada e de fácil manuseio.

Outra vantagem do método, é a uniformização e padronização que se vai adquirir nos programas, facilitando sobremaneira as futuras manutenções destes.

III - PLANEJAMENTO DOS TESTES DO PROGRAMA

O planejamento deve ser feito de maneira criteriosa e objetiva. Devem ser aproveitados os recursos disponíveis como ferramenta de apoio ao programador, sempre observando o uso otimizado do computador.

Um plano de teste bem elaborado, servirá de guia para o programador, quando da confecção dos arquivos e de todo complexo de testes que o programa irá precisar.

No entanto, este planejamento poderá não ser o definitivo, pois a medida que outros passos forem desenvolvidos, a assimilação aumentará, podendo surgir novas situações, que deverão ser incluídas no mesmo. Ao final da codificação este deverá ser obrigatoriamente revisto.

O plano de teste deverá incluir atividades como:

01. REVISÃO DO PROGRAMA

O primeiro procedimentos para assegurar a correção do programa é a revisão estática da codificação. Deverão portanto ser planejadas as revisões a serem efetuadas.

Existem vários processos de revisão, sendo os mais comuns, a revisão do código, o teste de lógica e a revisão formal, que é também conhecida como WALK-THROUGH.

A revisão do código e o teste de lógica, devem ser aplicados para todos os programas, e a revisão formal, apenas para os programas que a justifiquem, por tratar-se de um processo caro.

02. ETAPAS DE TESTES A SEREM EXECUTADAS

De acordo com os procedimentos dos programas, devemos planejar as etapas de testes necessários. Esta informação nos dirá quantos arquivos serão necessários para o teste do programa.

Para exemplificar, vamos tomar como base um programa atualizador de cadastro, considerando a existência de um arquivo movimento com apenas 1 registro de mesma identificação no cadastro. Teríamos as seguintes etapas:

Etapa 1 - Movimento com exclusões e alterações para arquivo cadastro vazio.

Etapa 2 - Movimento com inclusões para cadastro vazio.

Etapa 3 - Movimento com inclusões, alterações e exclusões corretas para cadastro criado na etapa 2.

Etapa 4 - Movimento com condições inválidas; inclusão para chave existente no cadastro criado na etapa 3.

Estas etapas mostram de uma maneira geral como podemos planejar as condições de testes, sendo que para cada procedimento teremos condições diferentes, e servirá como apoio ao planejamento dos arquivos a serem utilizados.

03. PLANEJAMENTO DOS ARQUIVOS DE ENTRADA

Planejar o conteúdo dos arquivos que foram identificados como necessários para o teste completo verificando o fator qualitativo e quantitativo dos mesmos.

Os arquivos podem ser planejados da seguinte maneira:

03.01. ARQUIVOS CORRETOS

Deve ser planejado para que satisfaça todas as condições de passagem correta de um programa. Os arquivos devem ser projetados, de acordo com sua definição devendo, em termos de conteúdo, se aproximar o máximo possível de um arquivo real que será utilizado em produção. Para tanto algumas condições devem ser lembradas:

- GERAL
  - . Arquivos com apenas um registro;
  - . Arquivos sem registro;
  - . Limites máximos e mínimos de conjuntos de registros;
  - . Registros de arquivos variáveis, com tamanho máximo, mínimo e intermediário.

- MERGE DE ARQUIVOS
  - . As identificações tenham condições de igualdade e desigualdade;
  - . Um arquivo termina primeiro e vice-versa;

. Todas as identificações de um arquivo iguais a dos outros bem como todos desiguais.

- CONSISTÊNCIA E COMPATIBILIDADE
  . Todos os campos deverão estar corretos;
  . As consistências cruzadas devem estar corretas;
  . Todos os fechamentos devem estar corretos.

- CÁLCULO
  . Todos os valores numéricos, prevendo valores mínimo, máximo e intermediários.

- ATUALIZAÇÃO
  . Código de atualização correto, contendo inclusões, exclusões e alterações sempre corretas.

- FORMATAÇAO
  . Todos os tipos de registros de erão estar corretos.

- RELATÓRIOS
  . Quantidade de registros que preencham mais de uma página;
  . Quantidade de registros que não preencham uma página;
  . Quantidade de registros que terminam exatamente no limite da página.

- TABELAS
  . Registros que testem os limites dos indexadores e subscritos do programa;
  . Registros que testem as tabelas internas do programa.

03.02. ARQUIVOS ERRADOS

Deve ser planejado para que force o programa a identificar o erro. Os arquivos devem ser projetados de maneira que o programa seja testado quanto a condições anormais e absurdas, que raramente ocorrem e que não foram previstas pelo sistema, causando o cancelamento de um programa. Ainda, estes arquivos devem conter erros previstos pelo programa, forçando a verificação dos testes do mesmo.

- GERAL
  . Ausência e duplicidade de HEADER's, TRAILLER'S, CAPA DE LOTE, REGISTRO DE FECHAMENTO e outros do mesmo nível;
  . Arquivos com apenas 1 registro;
  . Arquivos sem registros;
  . Erros de parâmetros informados.

- MERGE DE ARQUIVOS
  . Prever as condições de erros de combinação de identificações entre os vários arquivos;

. Identificação, fora de sequência.

- CONSISTÊNCIA E COMPATIBILIDADE
  . Deve ter todos os campos consistidos com o mais variado tipo de erro;
  . As consistências cruzadas devem estar incorretas;
  . Todos os fechamentos devem estar incorretos.

- ATUALIZAÇÃO
  . Código de atualização incorreto;
  . Exclusão e alteração para não existentes;
  . Inclusão para já existente.

- FORMATAÇÃO
  . Registros fora de sequência ou agrupados incorretamente.

03.03. ARQUIVOS MISTOS

Apesar do programa já ter sido planejado para executar estas funções, um arquivo que podemos chamar de misto, pode propiciar mais uma condição diferente de teste. Dependendo da situação, pode-se utilizar os arquivos anteriormente planejados, pois as condições de testes serão iguais aos descritos no item 03.01 e 03.02.

04. PLANEJAMENTO DOS RESULTADOS

Para cada etapa deverão ser esboçados todos os resultados esperados em função dos arquivos de entrada. Isto servirá para facilitar ao programador a conferência dos arquivos e relatórios de saída gerados pelo programa. Normalmente utiliza-se o processo de conferência visual, que é cansativo, o que prejudica ao longo dos testes a conferência de todas as condições. Planejando-se os resultados poderemos utilizar conferência automática o que, além de facilitar, dará ainda um grau de confiabilidade maior nos resultados finais.

05. DOCUMENTAÇÃO DO PLANO DE TESTE

Tão importante quanto fazer o plano de testes é que estes sejam documentados e arquivados de uma maneira que permitam sua utilização quando de uma manutenção futura no programa, sem necessidades de refazê-lo.

IV - CODIFICAÇÃO

Consiste em transformar em códigos, que possam ser interpretados pelo computador, a solução lógica do programa.

A codificação do programa deverá ser feita de forma parcial, que tem a particularidade de codificação e testes simultâneos, ou seja, codifica-se a rotina principal e testa seu funcionamento, quando estiver correto, passará aos testes dos vários grupos de rotinas individualmente, e só se

executando o próximo, quando o anterior estiver correto.

Esta prática, associada a uma necessária padronização às normas de codificação, tem a vantagem do programa se tornar menos cansativo, pois o programador variará constantemente seu serviço, além de que, as rotinas principais estarão exaustivamente testados, aumentando a confiabilidade do programa.

É importante que o programa seja codificado de maneira clara e explicativa, facilitando futuras manutenções.

V - REVISÃO

A revisão sempre é necessária quando se pensa em otimizar ou melhorar alguma coisa. Neste ponto devemos fazer a revisão de tudo que já temos de concreto, e no caso, o programa e o planejamento do teste. A revisão sempre amplia a visão do programador em torno do que está sendo feito, dando-lhe mais segurança e confiabilidade.

Apesar do programador ter feito o fluxo lógico do programa, e codificado de acordo com ele, pode ter passado alguma condição desapercebida, ou a maneira como seu programa foi escrito, pode não ter sido a melhor.Ainda há a possibilidade da existência de erros na transcrição que poderão influenciar a lógica, sem ter aparecido erro de sintaxe na compilação. A revisão além de ser um ótimo apoio para descobrir estes erros, economizará um bom número de testes que seriam necessários processar no computador. Propomos os seguintes procedimentos para revisar um programa:

01. REVISÃO DO CÓDIGO

É um método bastante informal que deve no entanto obedecer uma lista de procedimentos para se assegurar que não foi esquecido nada. É executado pelo programador com ou sem ajuda de um terceiro, a critério deste.

O programador de posse de uma listagem simples do programa, tentará encontrar inicialmente erros de codificação e transcrição confrontando o programa com os diagramas de estrutura lógica e as folhas de codificação, a fim de assegurar que o programa foi codificado de acordo com a lógica desenvolvida e transcrito de acordo com o que foi codificado.

02. TESTE DE LÓGICA

Consiste na prática de seguir a lógica do programa através do código, simulando registros que atendam as condições do programa. O uso de um terceiro é aconselhável, pois este poderá auxiliar bastante no acompanhamento de processamento dos registros no programa. Sendo isto na verdade um teste normal, devem ser testadas todas as condições possíveis, sempre em obediência ao plano de teste elaborado.

03. REVISÃO FORMAL

A revisão formal se caracteriza pela existência de um método e pelo desenvolvimento formal de outras pessoas no processo, ficando o revisado

praticamente em situação de expectador. Possui como grande vantagem sobre as revisões informais o fato de além de provar a correção dos programas, servir como instrumento de divulgação de conhecimentos entre o pessoal, bem como revisar constantemente a metodologia de desenvolvimento de programas através das idéias que surgem no trabalho de grupo.

Por ser um processo caro, nem sempre compensa uma revisão deste tipo, devendo ser observados critérios para determinar sua necessidade.

03.01. DETERMINAÇÃO DA NECESSIDADE

A revisão formal deve ser efetuada sempre que o programa possua procedimentos mais complexos ou numerosos, para os quais normalmente não conseguiríamos obter todas as condições de testes necessários, não devendo ser aplicada a programas pequenos ou fáceis, a menos que existam sérias limitações com relação ao uso do computador para teste.

A solicitação da revisão pode partir do programador que desenvolveu o programa, sua chefia ou de elementos diretamente ligados ao sistema em desenvolvimento.

03.02. PARTICIPANTES E SUAS RESPONSABILIDADES

Todos os participantes têm no processo, responsabilidades iguais perante a correção do programa, uma vez que as decisões devem ser tomadas em consenso pelo grupo. Todos devem estar familiarizados com o método e dispostos a ajudar.

Na escolha dos revisores deve ser incluído pelo menos, sempre que possível, um elemento que participa do mesmo projeto podendo ser programador ou analista, um programador experiente e um de menor experiência que não participem necessariamente do projeto.

Por uma questão de organização, definem-se três tipos de participantes:

- O Coordenador da Revisão, que é o responsável por coordenar o grupo, e que tem basicamente como atribuições:
  - . Determinar e avisar os participantes da data, hora e local para a revisão;
  - . Encaminhar o material necessário aos participantes;
  - . Coordenar a sessão, mantendo a ordem, orientando para que cada um fale de uma vez, manter a discussão rigorosamente sobre o assunto;
  - . Opinar e orientar o grupo para suas decisões.

- Os revisores, que devem participar do processo, obedecendo as determinações do Coordenador de Revisão, com o objetivo único de opinar acerca da matéria em questão. Deve haver um número entre 1 e 3 revisores, dependendo da complexidade do programa.

- O Revisado, que deve simplesmente assistir ao processo, sem direito a opinar, tendo como única atribuição corrigir os erros apontados pelos revisores.

03.03. PROCEDIMENTOS

O Coordenador da Revisão deve encaminhar com antecedência o material necessário aos participantes, que deve incluir a última compilação sem erros , diagrama de estrutura, lay-outs, etc, material este que cada participante deverá revisar sozinho antes da reunião, anotando os pontos que julgue errado ou obscuro.

A reunião deve ser conduzida de modo a que os participantes apontem os problemas, porém sem resolvê-los ou sugerir soluções, anotando-se cada observação aprovada em consenso pelo grupo. Deve ser realizada em local isolado onde não haja interrupções de espécie alguma, sendo desejável a existência de quadro, flip-chart ou outros meios de apoio. Sua demora não deverá exceder a duas horas.

Cada participante deve fazer ao final da revisão ao menos um comentário positivo e um negativo, sempre ao programa e nunca ao programador. Todas as partes do programa devem ser revisadas e na sua conclusão o Coordenador da Revisão deve preparar um relatório com o sumário das observações e entregar ao programador para o acerto do programa. Este relatório final da revisão deve ser claro, de maneira que qualquer pessoa possa entendê-lo sem ter participado da revisão, bem como deve ser arquivado com a documentação do programa.

A revisão pode acarretar em aprovação do programa ou solicitação de nova revisão. No último caso, após os acertos do programa, este deve ser novamente revisado, devendo-se porém, evitar repetição dos comentários feitos anteriormente.

04. LISTA DE VERIFICAÇÃO

Para se estabelecer uma linha de ação sobre os pontos a verificar dentro de um programa, é aconselhável que em cada empresa, de acordo com sua realidade e com as linguagens mais utilizadas, desenvolver seu próprio método. Como modelo, podemos citar para linguagem COBOL:

04.01. IDENTIFICAÇÃO DIVISION

. O nome do programa e quadro de explicações;
. Validade da função descrita para o programa.

04.02. ENVIRONMENT DIVISION

. Os nomes externos dos arquivos correspondem ao pedido;
. Ausência ou duplicidade de algum arquivo.

04.03. DATA DIVISION/FILE SECTION

. Existência de FD para arquivos comuns e SD para arquivos SORT;
. Correspondência com os nomes da SELECT;
. Presença da cláusula "RECORD CONTAINS" para forçar a verificação do tamanho do registro definido;
. As definições dos arquivos correspondem a sua característica física;

. Definição dos registros.

04.04. DATA DIVISION/WORKING-STORAGE SECTION

. Os acumuladores numéricos e inicializados;
. Caracter de controle válido nas linhas de impressão;
. Definição dos registros;
. Números de nível, para evitar que um item grupo abranja maior ou menor número de campos que o previsto;
. Tamanho de subscritos, comportam as ocorrências máximas;
. Literais estão escritos corretamente.

04.05. PROCEDURE DIVISION

. Parâmetros passados para as subrotinas chamadas pelo programa estão no formato e ordem requeridos;
. Existência de LINKAGE SECTION e PROCEDURE DIVISION USING... quando o programa recebe parâmetros;
. Existe um comando SET inicializando o indexador referente a tabela que se vai ser pesquisada, quando do uso de SEARCH sem opção ALL;
. Se não está sendo usado o indexador de uma tabela em outra;
. Validade dos campos usados em cálculos, inclusive validade de suas fórmulas;
. Todas as sections tem a palavra SECTION e terminam com EXIT;
. Todos os arquivos são abertos/fechados corretamente;
. Alinhamento de IF's procurando ausência ou excesso de pontos;
. A área receptora está correta na utilização do comando STRING;
. Tamanho de áreas do READ INTO e WRITE FROM, estão de acordo com o definido na FILE SECTION;
. DATA/HORA, quando necessários são obtidos como primeiras instruções;
. Parâmetros são validados quanto a erro ou ausência;
. O uso correto da palavra NOT em condições compostas;
. Em ninhos de comparação para cada IF existe o ELSE correspondente;
. Inexistência da possibilidade de leitura após fim de arquivo , inicialização e reinicialização corretos de contadores, acumuladores, tabelas e chaves;
. A última quebra do programa é efetuada na condição de término;
. O primeiro e último registro de qualquer arquivo é processado corretamente;
. GO TO para fora de sections;
. Referência a áreas de saída logo após um WRITE ou a áreas de entrada antes de um READ;
. Os literais estão escritos corretamente;
. Existem comandos que não serão executados;
. Existem DATA-NAMES que não estão sendo usados;

. Se o programa possui SORT interno:
OPEN-CLOSE e STOP RUN fora das sections da INPUT/OUTPUT PROCEDURE.
Os campos de classificação estão corretos (tamanho, posição,etc)
Existe desvio sem retorno para fora da INPUT/OUTPUT procedures.

VI - PREPARAÇÃO DOS ARQUIVOS

Na preparação física dos arquivos, deverão ser identificados quais os recursos disponíveis para sua criação bem como os meios de armazenamentos possíveis de utilizar. Dentre estes, o programador deve selecionar quais são os que melhor atendem as suas necessidades em função das características dos arquivos. Citamos alguns meios que podem ser utilizados:

01. CODIFICAÇÃO DE REGISTROS

Este recurso deve ser utilizado apenas quando o volume de dados é pequeno, pois ao contrário torna-se uma tarefa bastante estafante. Quando possível, deve ser feito diretamente utilizando-se documentos de entrada , que auxiliarão bastante, já que estes possuem formatos apropriados. Após sua codificação deverão ser transcritos para cartões ou meios magnéticos apropriados.

02. RECURSOS DE HARDWARE

Alguns computadores aceitam entrada e/ou alteração de dados diretamente por equipamento periférico. No caso de alimentação de dados deve ser aproveitado apenas para arquivos pequenos, pois é semelhante ao caso anterior. Já a alteração de dados de arquivos existentes, é uma prática aconselhável, pois o programador terá controle da alteração no momento que está processando. As limitações deste recurso devem ser avaliadas antes de optar pelo seu uso.

03. RECURSOS DE SOFTWARES

No mercado existe um bom número de softwares geradores de arquivos de testes. Eles tem demonstrado um bom grau de eficiência e apoio à programação, e podem ser processados juntos com o programa ou criando arquivos isoladamente. A pouca utilização destes softwares ocorre por falta de conhecimento do uso e potencialidade de condições que pode oferecer. Seu uso deve ser incentivado ao máximo, pois na maioria deles, a geração de um arquivo é rápida.

Praticamente todos os equipamentos já trazem um conjunto de utilitários que facilitam a tarefa de criação de arquivos.

04. PROGRAMAS ESPECIAIS

Sua utilização ocorre quando o arquivo é mais complexo e não pode ser gerado manualmente ou algum software disponível. Neste caso, codifica -

se um programa, normalmente elementar, que fará a criação de registros para o arquivo no formato desejado. Este programa, por ser temporário, não necessita seguir qualquer padronização e pode, por conseguinte ser escrito em pouco tempo.

05. O PRÓPRIO PROGRAMA

Este é um recurso que pode ser utilizado trazendo bons resultados. Após a leitura dos arquivos são transformados os registros lidos, formatando-os para as condições desejadas.

Na inexistência de um arquivo para formatar, é possível substituir os comandos de leitura por procedimentos que criem os registros com as condições desejadas.

06. COMBINAÇÃO DE RECURSOS

Muitas vezes pode ocorrer que um dos recursos existentes não consiga criar um arquivo conforme desejado. Neste caso podemos combinar o que dispomos, no sentido de que um determinado recurso crie um arquivo primitivo e outros o transformem para as condições necessárias a sua utilização no programa.

VII - EXECUÇÃO DOS TESTES

Consiste em processar o programa para cada etapa planejada, no sentido de apontar os erros de lógica. Se houver erro, o programa deve ser corrigido e executado novamente para esta condição. Só depois de se obter os resultados esperados é que a etapa seguinte deverá ser processada, até que a última seja executada.

01. TESTE PROGRESSIVO

O teste progressivo é o método que se aplica quando a codificação do programa é feita de forma parcial. Aqui, o teste se processa simultaneamente com a codificação do programa; primeiro codifica-se a estrutura principal de controle das rotinas, sem se preocupar com as rotinas de procedimentos específicos e testa-se a correção da lógica. Depois, os conjuntos de rotinas de procedimentos serão agregados ao programa nas etapas previstas no plano de teste. O importante é que somente se agregue ao programa um novo conjunto quando o que estiver em teste for considerado correto.

O que se obtém de vantagem com isto é que os erros poderão ser rapidamente localizados porque devem estar contidos apenas nas rotinas adicionadas, pois já sabe-se que as anteriores estão corretas.

Quando um programa é extenso e o programador codifica-o completamente, é normal que tanto sua produtividade quanto a previsão deminuam com o tempo por ser uma tarefa consativa. Por outro lado, alternando-se codificação e testes irão terminar estes problemas além do que haverá uma motivação maior por começar a obter resultados mais rápidos.

VIII - VERIFICAÇÃO DOS RESULTADOS

A escolha dos meios para verificação dos resultados, deve ser feita no sentido de facilitar ao programador a conferência das saídas geradas pelo programa. Como geralmente ocorre, estas conferências são feitas visualmente, sendo este processo bastante cansativo, nem sempre motivando o programador a conferir todas as condições testadas. Com ajuda de alguns recursos, estas conferências poderão ser efetuadas com um esforço humano menor , apresentando um grau de confiabilidade bem maior.

01. PRÉ-FORMATAÇÃO DE ARQUIVOS

A pré-formatação consiste em criar em um arquivo ou no papel, a imagem real do arquivo ou relatório de saída. Obtida como resultado do planejamento dos testes, este é o primeiro passo para a utilização de outros recursos, que permitirão uma melhor conferência visual ou automática, além de mostrar antecipadamente a imagem do resultado final do programa.

02. CONFERÊNCIA VISUAL

É a prática mais utilizada, sendo necessário apenas listar os arquivos e relatórios gerados pelo programa e conferir os resultados em função dos arquivos de entrada. Para cada execução de teste, todos os registros devem ser novamente conferidos, o que demanda um longo tempo. Para facilitar a conferência, o programador poderá fazer um programa listador dos arquivos , que vai separar seus campos, facilitando a identificação dos mesmos, ou até mesmo utilizar algum software com a mesma função. Se existe a pré-formatação dos arquivos, esta tarefa é bem mais rápida pois o único trabalho do programador é comparar o resultado do programa com o pré-formatado, sem se preocupar com o que o programa deveria processar.

03. CONFERÊNCIA AUTOMÁTICA

É necessário ter sido feita pré-formatação dos arquivos de saída e que esta tenha sido gravada em algum meio magnético. Com a utilização de programas especiais ou softwares de verificação quando disponíveis, compatibiliza-se os dois arquivos, o gerado pelo programa e o pré-formatado, emitindo um relatório que aponte todas as diferenças; se não houver, significa que o arquivo gerado está certo.

A melhor utilização deste método ocorre na proporção em que aumenta o volume dos dados gerados pelo programa em teste. Na prática é um recurso ainda pouco utilizado, porém pela economia que proporciona ao otimizar o tempo gasto pelo programador e pela confiabilidade que apresenta, certamente é um método que tende a se difundir.

04. FACILIDADES DAS LINGUAGENS

Nas linguagens de programação de alto nível, existem funções de apoio ao programador. Estas funções podem ser de grande auxílio, pois mostra em geral o andamento de um programa quando em execução. Temos comandos de DEBUG, bastante utilizados quando ocorre um erro mais difícil de identificar,

que consistem em mostrar os caminhos passados pelo programa. Outros que mostram quantas vezes cada comando foi executado, apresentam estatística que ajudam a otimizar o programa, mostrando onde ele gastou mais tempo. Dentro de cada linguagem em cada máquina, encontraremos estas facilidades, que devem ser exploradas e usadas, pois poderão ocorrer condições de erro de programa, que estas facilidades poderão auxiliar bastante na detecção dos mesmos.

CONSIDERAÇÕES FINAIS

E natural que a descoberta da necessidade de se utilizar de uma metodologia de desenvolvimento e testes de programas estruturados, venha a ocorrer quando a manutenção destes começa a crescer e tornar-se crítica, e , numa análise mais apurada das causas, concluímos que esta deve-se em parte ao fato dos programas terem sido mal desenvolvidos e testados.

Efetivamente, pode parecer a principio que este processo vem acarretar um custo maior no desenvolvimento, porque novos procedimentos estarão sendo acrescentados à atividade de programação. Por outro lado há de se observar que grande esforço empregado terá seu retorno já no desenvolvimento do próprio programa, pela produtividade maior que será alcançada na aplicação de um método objetivo.

Um outro aspecto a ser considerado é que o tempo decorrido para o desenvolvimento completo de um sistema deve diminuir sensivelmente porque para se obter o mesmo resultado em termos de qualidade e confiabilidade, através dos processos tradicionais, serão consumidos maiores recursos; nota-se que devemos considerar como recursos não só pessoal, como também equipamentos e materiais utilizados para desenvolver e testar os programas.

O retorno maior, porém, está localizado na fase de operação no sistema, onde teremos uma grande redução no tocante à ocorrência de problemas imprevistos.

Implantar esta metodologia, não é tarefa fácil, pois a adaptação do pessoal para a aplicação do método exige um trabalho de conscientização , com um alto grau de envolvimento, participação e motivação, no sentido de neutralizar as reações contrárias que surgem naturalmente quando ocorrem mudanças em uma organização.

B I B L I O G R A F I A

. METODOLOGIA CELEPAR - MC-DMS - CELEPAR

. MANUAL DE NORMAS E PROCEDIMENTOS DE PROGRAMAÇÃO CELEPAR

. A METODOLOGIA DE PROGRAMAÇÃO - JOSÉ DIDYK JUNIOR

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

UN SISTEMA DE PROGRAMACION AUTOMATICA

Leopoldo H. Carranza

RESUMEN

Un sistema de programación automática capaz de interpretar la descripción de un problema y sintetizar automáticamente un programa que lo resuelve es descripto brevemente.

Un lenguaje de especificaciones basado en el cálculo de predicados de la lógica matemática propuesto como interface con el usuario es detallado y su aplicación al manejo de base de datos es comentado.

Algunos aspectos teóricos sobre el poder expresivo del lenguaje y problemas de computabilidad son analizados informalmente.

UN SISTEMA DE PROGRAMACION AUTOMATICA

INTRODUCCION

Un sistema de programación automática capaz de producir automáticamente programas para resolver una clase restringida de problemas, es descripto brevemente.

A diferencia del enfoque habitual este sistema no requiere de los programas para probar teoremas, en cambio es una extensión del sistema de manejo de / base de datos.

La programación automática es la aplicación del computador a la producción de sus propios programas, esto es: usar el computador para mecanizar las tareas de programación. En cierto sentido es software para producir software.

El objetivo final es implementar un sistema capaz de entender los requerimientos del usuario, analizar el problema y usando su conocimiento sobre el tema, obtener el método de solución, codificar y optimizar el programa necesario.

Este objetivo está aún lejos de ser alcanzado y quedan aún muchas dificultades que vencer.

Una dificultad es que en general el problema puede ser recursivamente insoluble, esto es: no computable, o sea que no exista un programa que lo resuelva.

Sin embargo es posible aproximarse a este objetivo final.
La idea es restringir el universo del discurso, reduciendo la clase de problemas a aquellos que se puede asegurar que existe un programa que los resuelve y se conoce un método para obtener ese programa.

El sistema que aquí se describe se basa en limitar la clase de problemas a tratar restringiendo el lenguaje al universo de una base de datos.

En un trabajo anterior del autor se explicaban bajo otro punto de vista algunas de estas ideas.

El sistema propuesto está básicamente constituído por un compilador (de hecho un compilador de compilador) capaz de traducir el lenguaje de especificación al FORTRAN y se diferencia de otros tales como el de Z.Manna y R.Waldinger en que sus objetivos son más modestos como para no requerir que los programas para probar teoremas y es en cambio similar al Model II pero difiere en su concepción y en particular es más expresivo que los lenguajes de alto nivel de manejo de base de datos como el INGRES o el SQL (que no permiten obtener la clausura de una relación) y a diferencia de otros lenguajes como el SETL no requiere como primitivas las funciones de agregado.

Este sistema está parcialmente implementado y solo algunos componentes fueron probados. El proyecto fue encarado con fines de investigación y como parte de las tareas académicas del autor.

En lo que sigue se explican algunos conceptos básicos y se dan algunas definiciones preliminares y fundamentos teóricos, se describe el lenguaje de especificaciones y se comentan brevemente algunas de sus características más importantes.

Se omite toda referencia a los problemas de implementación, eficiencia, optimización, etc., así como toda referencia a la bibliografía clásica.

Se advierte que la exposición no es rigurosa ni formal.

CONCEPTOS BASICOS

Un sistema de programación automática es un sistema capaz de interpretar la descripción de un problema y sintetizar automáticamente un programa adecuado para resolver el problema planteado.

La descripción del problema puede realizarse a dos niveles: en términos de requerimientos o de especificaciones.

Los requerimientos son las condiciones que deben cumplir los resultados en función de los datos de entrada pero sin especificar el método de solución. No es posible diseñar un sistema para resolver cualquier problema dados los requerimientos ya que puede no existir un algoritmo de solución.

Las especificaciones en cambio son las condiciones que deben cumplir los resultados en función de los datos de entrada incluyendo la indicación del método de solución adoptado.

El objetivo es obtener un sistema que a partir de las especificaciones genere un programa cuyo output satisfaga las especificaciones.

La idea es expresar las especificaciones en el lenguaje del cálculo de predicados de la lógica matemática. Las condiciones que debe cumplir el output en función del imput se pueden expresar con una fórmula lógica: La solución es buscar los valores que hacen verdadera fórmula, esto es: satisfacer las especificaciones.

Este lenguaje de especificaciones puede considerarse como un lenguaje de programación de muy alto nivel, en el que solo se indican las características generales que debe cumplir la solución y omitiendo detalles sobre el proceso del cálculo permitiendo con pocas líneas de código resolver problemas complejos.

En camio, los lenguajes de programación tradicionales obligan al programador a codificar una serie de detalles del proceso de cálculo lo que constituye una tarea tediosa y sujeta al error.

Se estima que para una amplia gama de aplicaciones el uso de un lenguaje de especificaciones de este tipo puede incrementar la productividad de los programadores en una magnitud apreciable.

Otra idea interesante es considerar las bases de datos como única estructura de datos.

El programa generado por el sistema toma como imput una base de datos y produce como output la misma base de datos pero modificada. El usuario ingresa sus datos a la base y obtiene sus resultados consultando la base modificada, de esta forma el usuario solo debe aprender el lenguaje de interfase con la base de datos.

El programador solo necesita conocer el lenguaje de especificaciones (y un poco la lógica) para describir las condiciones que debe cumplir la base de datos modificada en función de la inicial.

En este sentido el lenguaje de especificaciones es también un lenguaje de manejo de base de datos.

El modelo de base de datos usado es el modelo relacional que asimila el conjunto de valores almacenados en la base a una relación en el sentido matemático.

El esquema de la base de datos provee los predicados básicos con los que se construyen las especificaciones.

Un predicado básico es un atributo ó nombre de relación del esquema de la base de datos y para un valor dado será verdadero si y solo si dicho valor aparece en la base, de lo contrario es falso. Las especificaciones se construyen como una fórmula bien formada a partir de los predicados básicos usando conectores y cuantificadores.

El programa producido por el sistema se limita a recorrer la base de datos (tantas veces como sea necesario) hasta encontrar el conjunto de valores que cumplen las especificaciones. Estos valores son, a su vez, almacenados en la base de datos.

Pero las fórmulas que expresan las especificaciones no pueden ser arbitrarias, es necesario introducir algunas restricciones. Las variables deben estar restringidas a un universo finito: solo pueden tomar valores entre los contenidos en la base o estar expresadas en función de estos valores.

Estas restricciones son necesarias para asegurar que el problema sea computable y reducirlo fundamentalmente a una simple búsqueda en la base de datos.

Estas ideas y otras han sido expuestas por el autor en otro trabajo, ya mencionado, en el que se hace énfasis en los aspectos teóricos de las bases de datos.

FUNDAMENTOS

Sea el lenguaje del cálculo de predicados L= (Lsim, Var, Oper, Pred) donde Lsimb. Var, Oper y Pred son conjuntos de símbolos lógicos (conectores y cuantificadores) variables, operadores y predicados respectivamente.

Con estos elementos definimos la sintáxis y semántica del lenguaje en la forma habitual.

SINTAXIS

Un término es una variable o expresión de la forma F (t1,t2,...,tn) donde F es un operador n-ádico y t1, t2,...,tn son términos.
Una constante es un operador cero-ádico.
Un predicado es una expresión de la forma P (t1,t2,...,tn) donde P es un símbolo de predicado n-ario y t1,t2,...,tn son términos o es de la forma (AvB), (A$\wedge$B),($\neg$A), (A=>B), (A$\Leftrightarrow$B), ($\forall$v,A), ($\exists$v,A) donde AyB predicados y v una variable.

SEMANTICA

Sea U= (Univ,Fun,Rel) una estructura donde Univ es un conjunto de elementos (universo del discurso), Fun un conjunto de funciones y Rel un conjunto de relaciones definidas entre los elementos de Univ.

Una interpretación o modelo del lenguaje L en la estructura U es una correspondencia que asigna a cada variable como rango el universo y a cada operador una función y a cada símbolo de predicado una relación.-

Si v es una variable la interpretación le asigna algún vi en Univ.-

Si F es un operador n-ádico la interpretación le asigna una función fi; $(Univ)^n \rightarrow Univ$.

Si P es un símbolo de predicado n-ario, la interpretación le asigna una relación $Ri \in (Univ)^n$.-

Si F (t1,t2,...,tn) es un término su denotación se define por:

$$i\,(f(t1,t2,\ldots,tn)) = fi(i(t1),\ i\,(t2),\ldots,i\,(tn))$$

(si f es 0-ádico, fi es una constante)

Si P (t1,t2,...,tn) es un predicado su denotación es:

$$i(\ P(t1,t2,\ldots,tn) = (i(t1),\ i(t2),\ldots,i(tn)) \in Ri$$

(Si P es 0-ario, su interpretación es un valor de verdad)

Si A y B son predicados y v una variable entonces:

i (AvB) = i(A) ó i (B)
i ($A \wedge B$) = i(A) y i (B)
i ($\neg A$) = no i (A)
i ($A \Rightarrow B$) = si i (A) entonces i (B)
i ($A \Leftrightarrow B$)= i (A) si y solo si i (B)
i ($\forall v, A$) = para todo vi, i (A)
i ($\exists v, A$)= para algún vi, i(A)

Con este procedimiento se asigna un significado a cada expresión (término o predicado) del lenguaje.

Un cálculo es un sistema formal C=(Lexpr,Axm,Inf) donde Lexpr es el conjunto de expresiones en un lenguaje L, Axm es un subconjunto de Lexpr elegido como axiomas e Inf es un conjunto de reglas de interferencia que a partir de ciertas secuencias de expresiones permite encontrar otras. -

Una expresión e es deducible (es un teorema) en el sistema bajo el conjunto de hipótesis T; $T \vdash e$, si hay una prueba de e, esto es si hay una secuencia finita de expresiones que a partir de los axiomas y las hipótesis (Axm$\cup$T) permite llegar a e por sucesivas aplicaciones de las reglas de inferencia.

Un conjunto T es consistente si para ninguna expresión e vale: $T \vdash e$ y $T \vdash (\neg e)$.

Una expresión e tiene un modelo en la estructura U bajo las hipótesis T; se escribe $T \models e$, si hay una interpretación que satisfaga las hipótesis y hacen verdadera la expresión e .-

Una expresión e es válida si es verdadera en toda posible interpretación sobre U. Se escribe $U \models e$.-

Una teoría T de una estructura U en un lenguaje L es el conjunto de expresiones de L que son válidas en U. -

Una teoría T es decidible si T es un conjunto recursivo. -

Con estas definiciones preliminares, se pueden tratar algunos aspectos teóricos importantes. -

Previamente suponemos que se haya hecho el trabajo de formalizar más rigurosamente lo anterior: elegir el conjunto de símbolos del lenguaje de modo que sea recursivo, con un conjunto numerable de variables, etc. y elegir los axiomas del cálculo de predicados de primer orden en la forma standard y fijar reglas de inferencia adecuadas, (por ej: modus pones) .-

Los siguientes teoremas pueden encontrarse en cualquier buen texto de lógica matemática (por ejemplo: los de J.D. Monk ó Z. Manna) .-

1 .- Los conjuntos de símbolos, expresiones, términos, predicados, axiomas lógicos y las pruebas lógicas a partir de un conjunto recursivo de hipótesis, son recursivos. -

2 .- El conjunto de teoremas lógicos es recursivamente enumerable, pero no recursivo. -

3 .- Teorema de completitud : T⊢e si y solo si T⊨e (en cálculo de primer orden) .-

Como resumen vale la pena notar estos teoremas famosos :

4 .- Hay sistemas formales (de deducción) completos para el cálculo de predicados de primer orden (pero no para el segundo orden) .-

5 .- El problema de validez de cálculo de predicados de primer orden es insoluble, pero parcialmente resoluble. -

6 .- La mayoría de las teorías interesantes (por ejemplo: la teoría de conjuntos, la aritmética de Peano) son indecidibles .-
Estos resultados implican la imposibilidad de obtener programas para resolver cualquier problema por lo que es necesario restringir la interpretación del lenguaje. -

BASE DE DATOS

Una base de datos relacional es BD = (Nrel,Natr,Esq,Int) donde Nrel es una colección de nombre de relaciones; Natr es un conjunto de nombres de atributos, Esq es un conjunto finito de expresiones de la forma Ri (Aj1, ..., Ajn) e Int es un conjunto de condiciones de integridad (expresiones en el lenguaje L) .-

Un estado S es una correspondencia que a cada nombre de relación le asigna una relación finita; esto es: un conjunto finito de n-uplas de valores, cada uno de sus componentes en el dominio del atributo correspondiente. -

Un estado es consistente si cumple con las condiciones de integridad.

Una computación es una secuencia de estados consistentes .

El dominio actual es el conjunto de valores realmente contenidos en la base en un estado. El dominio de definición es el conjunto de posibles valores ('a priori') . -

La idea es restringir nuestro lenguaje L= (Lsim,Oper,Pred) interpretando sus símbolos sobre una base de datos, o mejor: sobre una estructura U' =(Univ', Fun', Rel') donde Univ' se toma como la unión de los dominios de definición de los atributos, Fun' es una colección de funciones recursivas entre los elementos de ese dominio y Rel' el conjunto de relaciones contenidas en la base .

Se toma como símbolo de predicado los nombres de relación y su interpretación es la relación contenida en la base.

Si se limita el universo a la unión de dominios actuales (Valores realmente almacenados en la base en el estado actual), el universo resulta finito y la validez de cada expresión es decidible (basta recorrer la base tantas veces como necesaria, verificando para cada valor si satisface o nó la expresión dada). Pero una restricción de este tipo es demasiada limitativa para nuestros propósitos. Como se pretende actualizar la base, el universo debe incluír todos los valores posibles de ser almacenados en la vida útil de la base.

La idea es dejar el universo irrestricto pero entonces restringir la forma de las expresiones permitidas en el lenguaje, limitando las expresiones a las llamadas fórmulas ajustadas (tight) .-

Una fórmula está en forma normal disyuntiva si tiene la forma:

$$\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} e_{ij}$$

Esto es: si está expresada como una disyunción de conjunciones.

Una variable está ajustada en una de las conjunciones si:

1) Aparece por lo menos una vez en un predicado básico, no negado.
2) O aparece en una fórmula de la forma X= f(z1,z2,...zn) y z1,...zn están ajustadas.

Una fórmula está ajustada (o apretada?) si en la forma normal disyuntiva todas sus variables libres o ligadas están ajustadas.

Para decidir si una fórmula ajustada es válida, basta verificar la validez de la disyunción, esto es: investigar si se satisfacen alguna conjunción y como cada variable aparece por lo menos una vez en cada conjunción con un predicado básico no negado, o está expresada en función de variable ajustada, entonces basta recorrer los valores de la relación en la base de datos que la interpretación elegida asigna a los símbolos de predicado. Como las relaciones en la base son finitas este procedimiento siempre termina, por lo que se puede concluír que :

" La validez de una fórmula ajustable es decidible "

Con este resultado en mente, vemos de conectar todo lo anterior con el tema de la programación automática . -

LA PROGRAMACION AUTOMATICA

Estamos interesados en programas para resolver problemas. Se comienza formalizando un poco el concepto de problema. -

Sea T una teoría sobre una estructura U expresada en lenguaje L :

$$T=\{e:\ e \text{ es expresión en L y } U \models e\}$$

Un problema en la teoría T es un par P =(e,r) donde e es el enunciado: una expresión en el lenguaje L de la teoría y r es la respuesta pretendida .-

Un problema 'si-no' es un problema donde $r \in \{'si' , 'no'\}$

y se cumple (1) r= 'si' si y solo si $U \models e$
(2) r= 'no' si y solo si $U \not\models e$

Esto es: La respuesta será 'si' en caso que la expresión e sea una fórmula válida en la teoría y será 'no' en caso contrario.
Como las mayorías de las teorías interesantes son indecidibles (Cf.6) Se sigue que en general el problema 'si'-'no' es no computable.-

Como el problema de validez del cálculo de predicados de primer orden es insoluble, pero parcialmente resoluble, se sigue que en un problema del cálculo de predicado se puede hacer un programa que dada una expresión válida, obtenga en tiempo finito la respuesta 'si', en cambio si no es válida puede ciclar indefinidamente. -

Otra clase de problemas son los de forma P= (e(I,0),0) donde la entrada I y la salida 0, son variables con rango en el universo subadyacente de la teoría. El enunciado es: para los datos de entrada I encontrar los resultados 0 que satisfacen la especificación e (I,0) . Esto es encontrar los valores de 0 que verifican

$$U \models \forall I, \exists 0, e(I,0)$$

y por el teorema de completitud esto equivale a :

$$T \vdash \forall I, \exists 0, e(I,0)$$

lo que reduce el problema de la validez al de deducibilidad. -

Una prueba constructiva de este teorema da un método para encontar algún 0, para todo valor de I. La construcción automática de un programa que resuelva el problema se reduce a simular este método. Esta es la base de los sintetizadores de programas basados en probadores de teoremas (se aplica el pricipio de resolución y muchos esfuerzos se concentran en el perfeccionamiento del algoritmo de unificación). Pero los resultados sobre la indecibilidad de la mayoría de las teorías interesantes previene contra la adhesión a este enfoque tan general. Se ha seguido este trabajo bajo un criterio diferente, restringuiendo la forma e interpretación de las especificaciones y por lo tanto, la clase de problemas a tratar.

Lo que se busca es una función recursiva f que aplicada a los datos de entrada I, dé los valores de los resultados 0. Esto es: 0=f(I) de modo que e(I,0) (suponiendo una única respuesta). -
Un programa F es una descripción finita de f en algún lenguaje de programación .

Un sintetizador de programa es un programa Z que a partir de las especificaciones e produce como resultado el programa F que describe a f. Esto es: F =Z(e). Este programa Z, sintetizador de programas, puede verse como un compilador, que traduce el lenguaje de especificaciones L a un lenguaje de programación F (por ejemplo el FORTRAN), de modo que a cada posible ex

presión e en L le hace corresponder un programa F en F .-

Por supuesto, la condición es que cuando el programa F se corra con los datos I, se obtenga como resultado 0, los valores que hacen válida la expresión e (I,0). -
Con las restricciones anotadas, interpretando los símbolos de predicado como relaciones en una base de datos y limitándonos a usar fórmulas ajustadas, el problema es decidible, esto es: recursivamente resoluble.

EL SISTEMA

El sistema propuesto se reduce a un compilador (de hecho un compilador de compilador). Sus componentes principales son :

1) Un analizador lexicográfico: que se limita a detectar unos pocos símbolos ya que la mayor parte del trabajo de análisis se transfiere al parser.
2) Un analizador sintáctico: que aplica el método de Earley para hacer el parsing de gramáticas de contexto libre .
3) Un traductor propiamente dicho: que se basa en un esquema de traducción dirigido por la sintáxis.
4) Un traductor del código intermedio al Fortran.
5) Rutinas que simulan un sistema de manejo de base de datos.

El sistema toma como imput la gramática de la sintáxis y la gramática de la traducción y lee las especificaciones y produce un programa que a su vez lee los datos de una base de datos y almacena los resultados actualizando la misma base de datos. El programa obtenido se limita a recorrer los valores almacenados en la base hasta encontrar los que satisfacen las especificaciones.
Este procedimiento es simple, aunque poco eficiente. (se advirtió que no se han tenido en cuenta los problemas de optimización) . Se omiten más detalles sobre el sistema ya que la mayor parte de su diseño es standard y bien descripto en los textos de diseño de compiladores .-

El funcionamiento del sistema puede resumirse así:

1.- Lee las especificaciones y la lleva a la forma normal disyuntiva de prefijo (cambiando variables por variantes alfabéticas si es necesario).
2.- Para cada variable identifica los predicados básicos en que debe figurar en cada conjunción. Si una variable X no figura en algún predicado lógico en alguna conjunción debe localizar la expresión de la fórma X= f(Z1,...,ZN) en que figura. -
3.- Separa las variable en libre y ligadas. -
4.- Si X1,X2,...XN son las variables libres de la fórmula A genera el código intermedio:

```
for each X1 in U1,do
        :
        :
    for each Xn in Un,do
        if A then insert <X1,X2,...,Xn> in R
    od...od
```

que luego será traducido como un anidamiento de ciclos donde las variables X recorren las relaciones U asociadas contenidas en la base y cada N-upla de valores de las X que satisfacen la fórmula son insertadas en la base como una nueva relación R. -

5.-Si A es de la forma $\forall y,A(y)$ se las sustituye por el código:

```
for each in Uy,A(y)
```

si es de la forma $\exists y,A(y)$ se la cambia por

```
for any Y in Uy,A(y)
```

6.-Estas porciones de código se traducen luego generando ciclos que recorren los valores contenidos en los dominios asociados a cada variable para calcular el valor de verdad de la fórmula.

7. Es posible agregar pasos intermedios (para una mayor eficiencia) por ejemplo: modificar las variables que toman valor en el dominio de definición de los atributos, pueden cambiarse por otras cuyos rangos de valores son las N-upla de las ralaciones como en la práctica habitual en los lenguajes de consulta de base de datos. También es posible optimizar el código, pero estos pasos intermedios no han sido implementados. -

UNA EXTENSION

Tal como se ha descripto el lenguaje de especificaciones sufre de una serie de desventajas comunes a los lenguajes de consulta de base de datos: no es posible obtener la clausura transitiva de una relación.

Un remedio es extender el lenguaje así:

1.- En la sintáxis: Una ecuación es una expresión de la forma

A(X1,...,Xn) == E

donde A es una variable de predicado, X1,...,Xn son variables y E es un predicado, o mejor: una fórmula ajustada cuyas variables libres son: X1,...,Xn .-

2.- En la semántica:Se interpreta A(X1,...,Xn) como una relación R tal que <X1,...,Xn>$\in$R. sii X1,...,Xn hacen verdadera la fórmula E .-
Si la ecuación es recursiva, esto es: A==E(A) donde la variable de predicado figura en ambos miembros de la ecuación entonces se interpreta A como el punto fijo menos definido del funcional descripto por E .-

Otra extensión útil es permitir en la categoría de términos construcciones de la forma :

$$\{x:A(x)\}$$

donde A(x) debe ser una fórmula ajustada.

Con estas extensiones se incrementa el poder expresivo del lenguaje, que supera así al de los lenguajes de consulta de base de datos tradicionales. Se puede obtener la clausura transitiva de una relación y también las funciones de agregado tales como la sumatoria, el máximo y el mínimo de un conjunto sin necesidad de incluír estos operadores como primitivos del lenguaje.
Por último se debería agregar: una especificación es una secuencia de ecuaciones. Esto lleva al problema de estudiar una solución de un sistema de ecuaciones.

Por supuesto, siguiendo las técnicas de la semántica denotacional sería necesario exigir las condiciones de monotonía y continuidad a los funcionales para asegurar la unicidad del punto fijo menos definido, de modo que las ecuaciones recursivas tengan una interpretación aceptable.- Esta discusión está fuera de los límites de este trabajo. -

CONCLUSIONES Y OBSERVACIONES

Se ha descripto un sistema de programación automática como una mera extensión de los lenguajes de consulta de base de datos y se ha expuesto informalmente la teoría subadyacente.
El objetivo principal ha sido poner de manifiesto las dificultades más evidentes.
La restricción introducida de limitar las fórmulas aceptadas a las llamadas ajustadas asegura la existencia de un método de solución. -

Pero aún queda la pregunta: ¿ cuál es la clase más amplia de problemas para la que es posible encontrar siempre un método de solución ?

Se estima que las investigaciones en este sentido permitirán obtener sucesivas ampliaciones del poder expresivo del lenguaje de especificaciones.

Quedan en pié varias cuestiones y algunas ideas que se ha omitido a propósito, que vale la pena mencionar rápidamente :

1.- Como un usuario de un sistema de este tipo está interesado solo en respuestas de longitud finita, las relaciones en la base de datos pueden verse como fórmulas del tipo:

$$P(X) == x=a1 \vee x=a2 \vee \ldots \vee x=an$$

esto es como una disyunción de ecuaciones en la forma x=ai donde x es una variable y ai una constante.
En general se puede tomar todo predicado básico como una abreviatura de una disyunción de conjunciones de ecuaciones, así :

$$P(x1,\ldots,xn) == \bigvee_{j=1}^{m} \bigwedge_{i=1}^{n} xi = ai$$

2.- Aceptado lo anterior, una fórmula aceptable E es una función que a partir de expresiones de longitud finita como las indicadas, permite obtener otras del mismo tipo.

3.- Una especificación dada por la ecuación A==E es encontrar la fórmula de longitud finita expresada como disyunción de conjunciones de ecuaciones que satisface la ecuación dada.

4.- Con lo mencionado es posible una descripción más sintáctica del problema .-
Un sistema de programación automática como el propuesto puede entonces entenderse como un sistema que a partir de expresiones de longitud finita obtiene otras expresiones similares, en un tiempo finito .-

Se estima que esta línea de investigación, puede ayudar a mejorar nuestro entendimiento de los problemas de la programación automática, mientras se esperan los frutos de los estudios en Inteligencia Artificial. Las ideas expuestas son clásicas en las teorías de lenguajes formales, semántica denotacional, base de datos y teoría de la computación.-
La intención ha sido llamar la atención sobre este tema, mostrando un panorama que se estima de interés para todos, quienes estén involucrados en la tarea de desarrollo de software .-

* * * *

BIBLIOGRAFIA

1) J.Backus - Can Programming Be Liberated fron the Von Newman Style. A. Funtional Style Its Algebra of Programs - ACM - Comm.Vol 21,Num. 8, Agost 1978 .-

2) R.B. Banerji - Artificial Intelligence - North Holland - N.Y.1980.

3) M.A.Casanova y P.A. Bernstein- A Fourmal System for Reasoning about Programs Accesing a Relational Database- ACM,TOPLAS -Vol 2,Num.3, Jul 1980.

4) M.Davis- Computability and Unsolvalibity - Mc Graw Hill - 1958.

5) R.B.M. Dewar, A. Grand, SSU-Cheng Liu, y S.T. Schwarts - Programming by Refinement as Exemplified by the SETL Representation Sublanguage - ACM - TOPLAS, Vol 1 N 1, Julio 1979.

6) J.R. Hindley, B.Lerchery, S.P. Seldin -Introduction to Combinatory Logic - Cambridge Press 1972 .

7) D.W.Loveland - Automated Therem Proving: A Logical Basis - North Holland 1978 .

8) Z.Manna - Mathematical Theory of Computation Mc Graw Hill -1974.

9) Z.Manna - R.Waldinger - Studies on Automatic Programming Logic - North Holland - New York 1977 .

10) Z.Manna y R.Waldinger - A Deductive Approach to Programming Synthesis - ACM, TOPLAS, Vol.2 Num. 1, Enero 1980 .-

11) J.D.Monk - Mathematical Logic - Spring Verlag - 1976.

12) N.S.Paywes, A.Pnueli y S. Shastry -Use of a Non procedural Specification Language and Associated Program Generator in Software Development - ACM,TOPLAS . V.1 N 2,Oct. 1979.

13) R.Rusting et al.-Formal Semantic of Programming Languages -Prentice Hall 1972 .

14) J.E.Stoy -Denotational Semantics: The Scott - Strachey Approach to Programming Language Theory - MIT Press - 1979 -

15) D.A.Turner- A New Implementation Technique for Applicative Language- Software - Practice and Experience, Vol 9-31-49-1979.

16) D.H.D.Warren -Logic Programming and Compiler Wrinting - Software - Practice and Experience - Vol. 10, 97-125-1980.-

17) P.H. Winston - Artificial Intelligence - Addison Wesley - 1979 -

18) L.H.Carranza -Teoría de Base de Datos- 11 JAIIO - Oct.1980 - (Contiene otra referencias desde el punto de vista de base de datos.) .-

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

CAPITULO C

BASE DE DATOS

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

MUMPS, porque , quando e como usar?

Martín Tornquist
Divisao Acadêmica do CPD - UFRGS
Universidade Federal do Rio Grande do Sul
Avenida Oswaldo Aranha, 99
90000 Porto Alegre, RS, BRASIL

RESUMO

Este trabalho visa apresentar um sistema simples de gerência de banco de dados (SGBD), denominado MUMPS.A sua extrema simplicidade aliada ao seu poderio, faz pleno jus à tradicional citaçao de que "na simplicidade está a virtude" e temos certeza, interessará a muitas pessoas à procura de una ferramenta mais adequada à manipulaçao de suas informaçoes.

E descrita a evoluçao histórica do desenvolvimento de técnicas de programaçao e dos SGBD, caracterizando bem as suas origens e as lacunas que posteriormente vieram a preencher. Segue-se uma análise de beneficios e ou desvantagens obtidas pelo emprego dos mesmos.

Após sao brevemente abordados os temas centralizaçao e distribuiçao de sistemas de informaçao, definido-se o ambiente típico de utilizaçao da linguagem MUMPS. Varias aplicaçoes também sao citadas.

A linguagem MUMPS é apresentada nas suas características e potencialidades, mostrando claramente os benefícios obteníveis pela utilizaçao da mesma. Algunas aplicaçoes em MUMPS, desenvolvidas na UFRGS, sao apresentadas e comentadas.

OBJETIVOS:

Este trabalho visa apresentar um sistema simples de gerência de banco de dados (SGBD), denominado MUMPS.A sua extrema simplicidade aliada ao seu poderio,faz pleno jus à tradicional citaçao de que "na simplicidade está a virtude" e temos certeza, interessará a muitas pessoas à procura de uma ferramenta mais adequada à manipulaçao de suas informaçoes.

Para isso porém, serao necessários primeiro alguns comentários críticos sobre a evoluçao dos SGBD e seu uso.

INTRODUÇÃO

No mundo atual,especificamente na área de sistemas de informação baseados em processamento eletrônico de dados,a situação geral, do ponto de vista da qualidade dos sistemas que produzimos, nos parece bastante caótica. Podemos afirmar que a nossa profissão ainda não alcançou sua plena maturidade,maturidade esta, exigida pela complexidade dos sistemas que criamos.

Perplexos? Irritados? Pois bem,quantos dos nossos leitores se arriscariam a continuar a voar,se soubessem que o nível de qualidade e ou manutenção de uma dada companhia aérea, fosse equivalente ao nível de qualidade e ou confiabilidade dos sistemas por nós produzidos ? Nós ,certamente que não!

Se analisarmos a evolução das técnicas de desenvolvimento de programas nos últimos 30 anos, observaremos alguns fatos insólitos:

- crescimento assustador do Hardware disponível para o desenvolvimento de sistemas.

- proliferação de linguagens de programação, criando uma verdadeira torre de Babel.

- desenvolvimento de um verdadeiro arsenal de programas utilitários para auxiliar em todas as fases do desenvolvimento de sistemas.

- desenvolvimento e uso de um número crescente de técnicas de desenvolvimento de sistemas.O maior enfoque foi concentrado sobre técnicas repressivas e é de se estranhar que , somente nos últimos oito anos,tenham surgido estudos sobre os aspectos psicológicos da tarefa de programação.

Analisando esta evolução, alguns aspectos saltam à vista imediatamente:

- à exceção do aumento da velocidade e capacidade de armazenamento e à exceção da diminuição da largura e altura dos computadores a sua capacidade computacional inerente, continua a mesma. É espantoso que em uma área tão dinâmica e nova,ainda estejamos amarrados a uma mesma arquitetura,datada de 30 anos atrás.Podemos garantir-lhes que não é somente por falta de melhores opções !

- a nossa ferramenta básica para o desenvolvimento de sistemas de informação,a linguagem de programação,também não evolui muito.Para uma grande maioria,terminou com FORTRAN e COBOL!

- uma boa técnica de programação,tem sido tradicionalmente avaliada pelo tempo de codificação e teste dos programas,pela velocidade de execução,pela eficiência do código objeto gerado,pela clareza do

programa,pela facilidade e custo de manutenção e pela clareza da documentação. Poucos ,porém,se preocuparam com a satisfação do usuário do sistema.Vemos o usuário como parte integrante do processo de desenvolvimento de um sistema,e isto, ao nosso ver,tem sido negligenciado.

- como a nossa capacidade computacional não se alterou,continuamos sem poder resolver os mesmos problemas (ditos não computáveis) de 3 décadas atrás.

- o equilíbrio Hardware-Software está definitivamente pendendo para o lado do Software,numa relação que,segundo alguns especialistas,chegará a 9/1 em 1985 em termos de custo dos sistemas.

- o programador como ser humano , foi largamente desprezado,submergindo sob um mar de ferramentas e técnicas ,que teoricamente,pretendiam ser a solução dos seus problemas.Não conseguimos fugir da idéia,que o programador típico de hoje,perde um tempo excessivo com a manipulação e aprendizado de técnicas e linguagens de apoio, quando as maiores soluções para os atuais problemas estão dentro do programador e ou analista. Não podemos esquecer que , a nossa profissão consiste literalmente em montar castelos no ar.Poucas são as profissões que apresentam esta característica de extrema abstração. Sobre este último aspecto, Brooks já dizia : "The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds castles in air, from air, creating by the exertion of the imagination. Few media of creation are so easy to polish and rework, so readily capable of realizing grand conceptual structures. (As we shall see later, this very tractability has its own problems.)" (Brooks,1975) As ferramentas disponíveis hoje,pouco se preocupam com este fato e , por isso mesmo,falham!O maior problema hoje não é a falta de técnicas,é a falta de posicionamento individual,de grupo e de educação. Acreditamos que muitas das atuais técnicas de programação têm nivelado por baixo,restringindo uma característica exclusivamente humana e rara: a criatividade.

EVOLUÇÃO DOS SGBD

Os anos 60 presenciaram à centralização do processamento de dados,ditada por economias de escala.As necessidades dos anos 70,por sua vez ,forçaram a criação de sistema ON-LINE de acesso direto. Linguagens,sistemas operacionais e métodos de acesso às informações , têm profundos efeitos sobre programadores,gerentes e usuários.Gerenciar o trabalho de programaçao em um ambiente de rápidas e constantes mudanças,certamente não é tarefa simples.

Foi neste ambiente e com esta herança que os SGBD surgiram como a resposta a todos os problemas acima citados: centralização,acesso de informações ON-LINE e padronização de estruturas complexas.

Antecipando-nos a sua descrição , MUMPS já nasceu em um ambiente de acesso direto,ON-LINE, a um número reduzido de usuários(2 a 32).MUMPS não foi utilizado em processamento BATCH e as economias de escala não ditavam a sua centralização.

Sobre SGBD , uma palestra do Bachman marcou época:

"Just as the ancient viewed the Earth with the Sun revolving around it , so have the ancients of our information systems viewed a tab machine or computer with a sequential file flowing through it ... Each was an accurate model for its time and place ... This revolution in thinking is changing the programmer from a stationary viewer of objects passing before him in core into a mobile navigator who is able to probe and traverse a data base at will." (Bachman , 1973)

Um típico SGBD (Date , 1973) apresenta cinco componentes principais:

a) uma linguagem hospedeira,tipicamente uma linguagem de uso geral como COBOL ou PL/1.

b) uma linguagem de definição de dados,utilizada para a especificação da forma de armazenamento e do método de acesso aos dados.

c) um modelo de dados,que controla a informação que o usuário vê.

d) um sub-modelo de dados,que controla a parte das informações vista por um usuário individual.

e) um sistema de acesso direto,tipicamente um componente do S.O. empregado.

Date define uma série de vantagens a serem obtidas pelo uso do SGBD:

a) diminuição da redundância dos dados.

b) possível diminuição da inconsistência dos dados armazenados no BD.

c) compartilhamento de dados entre diversas aplicações.

d) facilidades para criar e manter padronizações.

e) facilidades para a aplicação de medidas de segurança.

f) manutenção da integridade física e lógica do BD.

g) independência dos dados.

h) disponibilidade de complexas estruturas de armazenamento e recuperação de dados.

A independência dos dados é citada como uma das maiores realizações dos SGBD. Date a definiu como "imunity of applications to change in storage strucures and access strategy". (Date,1975) A representação física dos dados poderá , portanto , ser modificada,sem influir no Software já desenvolvido sobre os mesmos dados.

Os SGBD por si só ,poderão gerar uma dependência procedural,isto é, a manutenção das hierarquias e redes do SGBD independendentemente de seu conteúdo. Os SGBD substituem dependência de dados por independência de dados.Ainda não sabemos se isto realmente é de real benefício.A maioria dos SGBD existentes,não conseguiram criar uma independência total da base de dados.

CUSTOS DE UM SGBD

Os custos de aquisição ou aluguel dos SGBD no mercado são elevadíssimos.Pior ainda,os custos de manutenção e Hardware do sistema,rapidamente superam os custos do SGBD propriamente dito. A necessidade de maior treinamento dos programadores,o tempo de processamento BATCH diminuído,o aumento dos custos de manutenção de arquivos e a maior complexidade geral do sistema,são outros fatores de custo menos aparentes.

COMPLEXIDADE DE UM SGBD

Um SGBD típico é um pacote de Software extremamente complexo,requerendo normalmente enormes quantidades de memória para uma boa performance. A complexidade de ajustar o Software ao aumento do número de usuários,cresce mais que proporcionalmente. A performance é altamente dependente da atuação do administrador do BD e de sua equipe.

Um erro em um SGBD frequentemente atravessa o domínio de cinco a seis linguagens,descritas em milhares de páginas de documentação.

IMPACTO NA ORGANIZAÇÃO

Muitas companhias instalaram um SGBD na ingênua esperança de grande e gloriosa centralização de informações.Para este estado grandemente contribuíram as tão propaladas SIG (Sistemas de Informação Gerenciais). A consolidação de arquivos e procedimentos de diversos departamentos é uma tarefa hercúlea em qualquer organização estabelecida.Cada departamento possui seus sistemas manuais estabelecidos e procedimentos próprios no uso do computador.Tentar removê-los , dentro de um departamento , é difícil,que dirá tentar integrá-los a nível de organização ! O pior é que , é perfeitamente possível que,junto à entrada em funcionamento do SGBD,surjam novos procedimentos manuais paralelos para atender às omissões e novas necessidades.

Muitos profissionais de PED vêem nos SGBD um fim em si e que a última tecnologia irá trazer a solução para todos os seus problemas criando "bons" sistemas. O SGBD , porém, força um fluxo de informações estranho ao departamento em nome da organização.Isto certamente será duro de engolir para muitos gerentes. Em face destes problemas,o enfoque de distribuição da capacidade computacional ganha novo ímpeto.Cada departamento ,ou departamentos , que dividam as mesmas informações,seguiriam seus próprios caminhos de computação. As prioridades e o andamento dos trabalhos seriam então geridas pelas normas tradicionais da gerência. Isto não é uma volta ao estado caótico de descentralização de 1960 ! As economias de escala oferecidas por computadores de grande porte,além de não serem mais necessariamente verdadeiras,são denegridas pela degradação do tempo de resposta e prontidão do sistema. Novas tecnologias,como MUMPS,se adaptam perfeitamente a este tipo de ambiente.

Apesar de seu custo, o mercado dos SGBD está crescendo,portanto está satisfazendo necessidades do mundo real ! Quais seriam estas?

a) estruturas de acesso direto eficientes e poderosas.

b) capacidade de comunicação ON-LINE.

c) capacidade de controle e segurança dos dados.

d) compatibilidade com o Software e Hardware existentes.

Logo , o sucesso dos SGBD reside basicamente nestas lacunas que preencheu. Conseguir ligar um /360 a vários terminais já foi um façanha em si , pequena , porém , se comparada com a adição das complexas estruturas de acesso de dados que os SGBD oferecem. Muitos usuários de SGBD optaram pelos seus sistemas apenas por estes dois fatores.

CENTRALIZAÇÃO x DISTRIBUIÇÃO

Os SGBD evoluíram das necessidades do processamento centralizado . MUMPS , por sua vez,sempre foi orientado para mini-computadores. Como as diferenças de performance entre os minis , os antigos midis e maxis está diminuindo cada vez mais, nem a economia de escala será mais necessariamente válida. Existe até quem já propôs uma alteração da velha lei de Grosch,afirmado que: "A performance de um computador é inversamente proporcional ao quadrado de seu preço" (Adams , 1962).

O usuário está ficando cada vez mais exigente (com razão) e sofisticado nas suas necessidades, o computador está por sua vez se tornando cada vez mais acessível e a programação menos esotérica. A maioria dos departamentos de uma empresa vai querer ter seu computador. Wagner criou o seu famoso "princípio da descentralização" , enunciado como : "If an organizational group , smaller than 30 people , requires computer assistance , it is better for the total enterprise that those people have exclusive use of their own computer - provided that the computer , big enough to do the job properly , will be loaded to over 10% of its capacity." (Wagner , 1976) . Os usuários gerenciarão os seus próprios recursos,estabelecerão suas próprias prioridades e manipularão suas informações.A maioria das mesmas serão intra-departamentais e o compartilhamento de informações será geralmente a exceção,não a regra. Na próxima década assitiremos à integração cada vez maior do computador nas empresas.

Em suma, os usuários estão sentindo as seguintes possibilidades e ou vantagens no uso de um computador próprio:

a) computação a custo baixo e previsível.

b) liberdade total de desenvolvimento.

c) facilidade de programação.

d) performance relativa melhorada.

e) potencial de crescimento e flexibilidade maiores.

f) ausência de controles e restrições externas.

g) grande motivação do usuário.

h) sistemas mais simples, portanto geralmente mais confiáveis.

i) menor tempo de resposta em aplicações interativas.

MUMPS é adequado a este enfoque,entretanto a parafernália administrativa que , normalmente está associada a um SGBD, não necessariamente se aplica a um ambiente MUMPS. O SGBD tem que ser suficientemente generalizado para atender a uma vasta gama de métodos , estruturas de acesso e interfaces com as linguagens hospedeiras. Muitas destas funções generalizadas dos SGBD são incorporados ao MUMPS sob a forma de funções e comandos simples. O programador tem ao seu dispor um completo conjunto destas funções,de fácil uso e entendimento. Seu trabalho será o de combiná-los adequadamente para uma determinada aplicação.Se necessitar de alguma estrutura de acesso peculiar ou de alguma organização de arquivos ausente naquelas primitivamente definidas no MUMPS, ele as criará fácilmente.

Os SGBD representam a evolução natural dos antigos sistemas em BATCH. Portanto o problema de compatibildade é bastante crítico. MUMPS por sua vez , apenas requer que seja compatível com ele mesmo.

O potencial do MUMPS assustaria a muito gerente de PED.Para esses , o conceito de uma equipe de programação , se restringe a grupos de programadores COBOL,trabalhando em cima de folhas de codificação,protegidos de realizarem grandes estragos pelo S.O. e pelo SGBD. Programadores de sistemas,analistas,operadores e o administrador do banco de dados,todos,têm papéis comuns ao do programador. Para o programador MUMPS, esta estrutura parece opressiva.Ele desempenha o papel de todas as especialidades acima-citadas.Ele pode em questão de horas , realizar tarefas,que levariam semanas em SGBD convencionais.A natureza interativa da linguagem normalmente leva o programador MUMPS a um profundo conhecimento de sua ferramenta. Esta flexibilidade tem seus problemas.Na pressa de terminar o programa , ou por motivos outros , o programador poderá utilizar truques ou simplificações. Os atuais programadores foram treinados a escrever programas "eficientes" . Truques de programação,quanto mais escuros melhor,são fatos corriqueiros no nosso dia a dia. Na verdade é uma característica inerente a todos nós.Qualquer linguagem ou SGBD está sujeita aos seus malefícios. Pensem no político que consegue falar o melhor português por meia hora e não chegar a conclusão alguma. A única maneira de eliminar estes truques é através do esforço consciente do programador.Para isto ele deverá ser exposto às boas técnicas e a boas razões para não utilizar aquelas consideradas negativas.

CARACTERÍSTICAS DO MUMPS

Este trabalho foi introduzido com o intuito de mostrar àqueles que se interessaram pela primeira parte, algumas das principais características do MUMPS, tecendo ainda alguns comentários sobre a nossa experiência com a linguagem.

MUMPS é um acrônimo para Massachussets General Hospital Utility MultiProgramming System. O sistema foi desenvolvido em 1972 e cresceu de baixo para cima. Isto é, trata-se de um sistema criado por usuários, para usuários e portanto de domínio público. Nenhum fabricante foi responsável pela sua criação ! O entusiasmo despertado pela linguagem, fez com que ele rapidamente se espalhasse e fosse oficialmente reconhecido um STANDARD MUMPS, pela ANSI (American National Standards Institute), em 1975. A ANSI já reconheceu quatro linguagens: FORTRAN, COBOL, MUMPS e BASIC (por ordem cronológica) e atualmente está estudando a inclusão da linguagem PASCAL). A linguagem é mantida pela ANSI e por grupos de entusiastas MUMPS chamados MUG's (MUMPS USER GROUP's). No Brasil, o MUG-BRASIL existe desde outubro de 1980. A maioria dos fabricantes (inclusive de computadores de grande porte como o B-6700 e /370) já oferecem versões do MUMPS nos seus equipamentos.

MUMPS tradicionalmente tem sido implementado em mini-computadores dedicados e nestes , age como sistema operacional. Neste papel, MUMPS desempenha as seguintes funções:

a) implementa um sistema de TIME-SHARING, para atendimento de "n" usuários, por terminal.

b) controla todo o tráfego de mensagens entre os terminais-UCP-demais periféricos.

c) implementa rotinas que permitem a criação, deleção, inserção e recuperação de dados em uma base de dados de modelo hierárquico. MUMPS chama esta estrutura de GLOBAL, pois todos os seus usuários poderão acessá-la simultaneamente.

d) implementa um interpretador de uma linguagem também chamada MUMPS.

Pelo acima exposto , fica claro que MUMPS é um sistema dedicado e orientado para aplicações conversacionais (interativas) de entrada e ou recuperação de informações (manipulação de textos), segundo um modelo hierárquico. O modelo de representação de dados hierárquico ,apesar de ser possível mostrar que não se adapta a todas as situações da vida real, atende à grande maioria das mesmas. Basta lembrar que o sistema IMS, que segue um modelo hierárquico, é um dos SGBD mais utilizados no mundo .

A linguagem MUMPS é uma linguagem procedural de alto nível que incorpora os habituais comandos de controle de fluxo de programas das linguagens mais tradicionais. O seu aprendizado é extremamente rápido, pois como a linguagem é interpretativa, a interação do programador com a linguagem pelo terminal, traz excelentes frutos a curto prazo. A nossa experiência tem mostrado que programadores aprendem a linguagem em duas semanas , enquanto que leigos precisam um pouco mais de um mês para dominá-la. Talvez tenha passado despercebido,mas como MUMPS normalmente está implementado em uma máquina dedicada, o programador apenas necessitará aprender UMA única linguagem.Não é por nada que se fala em uma "máquina MUMPS".

A linguagem apenas reconhece um tipo de dado, o "string" de comprimento variável e oferece uma ampla gama de operadores e funções para a sua manipulação,por conteúdo e formato ("pattern matching"). Esta capacidade de manipulação de caracteres , torna MUMPS uma ferramenta ideal para aplicações em escritórios,laboratórios,pequenas e médias empresas,bibliotecas,uso próprio , CAI's e CAD's ...etc. Não é de se estranhar que MUMPS domine completamente o cenário da computação na área médica nos EUA, seu país de origem.

MUMPS incorpora diretamente a capacidade de manipulação de estruturas hierárquicas. Poderão existir quantas arvores o programador desejar e estas serão esparsas , significando que os únicos nodos criados são aqueles aos quais foram explicitamente atribuídos valores , ou aqueles necessários para criar os caminhos de acesso da raíz aos novos nodos. Esta generalidade (que se estende às variáveis subscritas locais) , permite ao programador a utilização desta estrutura com os índices peculiares a uma dada aplicação.Algumas implementações de MUMPS aceitam "strings" como subscrito, permitindo , portanto , acesso por conteúdo. É , porém , importante lembrar , que o grau de generalidade obtido pelo MUMPS na sua estrutura hierárquica é um compromisso entre espaço e tempo, se comparada com um sistema no qual os subscritos são ordenados linearmente. Sendo o MUMPS um sistema de múltiplos usuários,que permite o acesso simultâneo dos mesmos , às estruturas hierárquicas ("Database sharing"), existem formas de garantir o acesso mutuamente exclusivo às mesmas, para evitar a perda de dados e ou abraços mortais ("DEADLOCK's").

A linguagem procura a maior independência dos dados possível e por isso mesmo não possui declarações de tipo algum.A facilidade de uso e conversão de construções que , em outras linguagens seriam representadas por vários tipos diferentes, possivelmente conflitantes e sem regra de conversão definida,realmente liberam o programador MUMPS de quaisquer restrições de máquina,permitindo a sua concentração total na solução do problema que estiver tratando !

Estas características da linguagem trazem consigo duas importantes consequências:

a) livre criatividade do programador, independente da máquina.

b) alta receptividade e entusiasmo pela linguagem.

Estes dois aspectos, aliados às características da linguagem, geram resultados surpreendentes a curto prazo !

Afirmamos , sem sombra de dúvida, que o desenvolvimento de sistemas em MUMPS, se comparados com o desenvolvimento de sistemas similares em outras linguagens, traz consigo as seguintes vantagens:

a) redução do tempo de desenvolvimento , de meses para semanas e de semanas para dias.

b) alta receptividade pelo usuário, devido à flexibilidade e manutentabilidade do sistema.

c) tempo de manutenção drasticamente reduzido (uma das características do MUMPS é a extrema facilidade de alterar programas).

d) maior confiabilidade devido a:

- estrutura modular forçada da linguagem.
- tamanho reduzido de cada rotina (10 a 40 linhas).
- grande motivação pela linguagem, normalmente encontrada nos seus programadores.

e) diminuição da fase de teste e depuração dos sistema, pelas facilidades oferecidas por um interpretador.

Gostaria de mostrar um exemplo da facilidade de teste e depuração que o MUMPS oferece: suponhamos que um programa MUMPS, que já estava processando a 15 minutos, de repente pára por algum motivo ! O programador MUMPS poderá imediatamente visualizar o estado de toda a sua base de dados, corrigí-la se necessário , editar o programa , se necessário , e continuar a execução a partir do ponto de parada. Parece-me que não será necessário comparar esta técnica com a vigente em CPD's que empregam linguagens convencionais !

O sistema MUMPS também apresenta algumas restrições, que deverão ser cuidadosamente analisadas antes de partirmos para a sua utilização.

- a plena capacidade da linguagem só é atingida quando implementada em máquina dedicada , normalmente um mini-computador. Quando sob a égide de um S.O. de uso geral, a linguagem geralmente se apresenta com algumas restrições, principalmente em termos de tempo de resposta.

- por ser uma linguagem interpretativa, o seu calcanhar de Aquilles é um possível tempo de resposta desastroso. Interpretadores já fizeram grande sucesso na década de 60 mas foram abandonados em favor dos compiladores, devido a este problema ! Porém, com os recentes avanços da tecnologia, que anunciam microprocessadores de 32 bits para 1981 a preços irrisórios, a tendência reverterá em benefício dos interpretadores , pois nenhum compilador poderá jamais oferecer a flexibilidade de um interpretador. Cabe aqui lembrar que MUMPS se destina a aplicações "IO-BOUND" e não àquelas que são "CPU-BOUND". Como nas primeiras , o tempo de transmissão e ou recepção e o tempo de interação do operador no terminal são enormes, se comparados com um ciclo da UCP, a relativa lentidão de um interpretador (quando comparada a um compilador) não se deverá fazer sentir.

- a linguagem somente poderá ser aproveitada em todo o seu potencial por bons programadores. MUMPS é uma linguagem elitista até certo ponto. Por não apresentar restrições alguma ao programador, os resultados dependerão essencialmente de sua conscientização da função de programação.

- o uso intenso de programação interativa permite que surja a figura do programador que desenvolve,programa e testa os seus sistemas ON-LINE, de preferência ao mesmo tempo ! Para evitar este tipo de elemento, a gerência da programação deverá tomar medidas cabíveis.

APLICAÇÕES

A UFRGS (Universidade Federal do Rio Grande do SUL) utiliza um sistema MUMPS no seu equipamento B-6700, desde julho de 1979. O seu maior usuário é O Núcleo de Informática Médica,grandemente impulsionado pela utilização do MUMPS.

Lideramos um pequeno grupo de pesquisa que tem desenvolvido algumas aplicações em MUMPS,com o objetivo-mor de testar a sua real capacidade. O primeiro projeto foi o desenvolvimento de um cross-compilador da linguagem PASCAL (BYTE , Setembro , 1978), para o computador HP-2100.Neste projeto foram gastos um total de 270 horas, da concepção à entrega do sistema. O programa é composto de 50 rotinas de em média 8 linhas, num surpreendente total de 480 linhas ! O mesmo projeto foi realizado na linguagem ALGOL, sendo que levou o dobro do tempo para a sua realização e gerou um programa de quase 3000 linhas !

Com o objetivo de testar os efeitos da linguagem com programadores que a desconheciam, foi oferecido um curso de 20 horas sobre MUMPS. No decorrer do curso foi entregue a definição de um sistema de recuperação de informações bibliográficas, para cada um dos integrantes do mesmo. A definição do sistema previa quatro módulos , cadastramento , listagem , atualização e recuperação das seguintes informações: autor,obra,ementa,palavras chave,código da obra e comentários livres associados a cada obra.Foi ainda definida uma linguagem de acesso a estas informações, que permitia o emprego de operadores booleanos e relacionais. Para medir o grau de entusiasmo despertado pela linguagem, o trabalho não foi caracterizado como sendo de entrega obrigatória. Ao cabo de duas semanas após o curso, o primeiro aluno entregou seu trabalho.O sistema completo , funcionando , tinha 130 linhas !! (a listagem ocupava apenas duas folhas). De um total de oito alunos, três alunos terminaram o projeto com resultados similares dentro de um mês após terem aprendido a linguagem.Os demais apenas o concluíram parcialmente, alegando falta de tempo. Consideramos os resultados obtidos como altamente positivos e encorajadores.

O terceiro projeto, de codinome DIÁLOGO , visa estudar as capacidades do MUMPS na criação de diálogos em linguagem semi-natural. Além da entrada e ou recuperação de informações, este sistema se prestará a uma futura inclusão em um sistema de avaliação de alunos por terminal, onde permitiria respostas em aberto e a manutenção de um diálogo com os alunos. Os resultados obtidos até o momento são altamente estimulantes !

CONCLUSÃO:

O MUMPS , contrariamente a muitos sistemas , acredita e investe na capacidade e conscientização de seus programadores . O sistema , apesar de ser voltado para aplicações conversacionais de recuperação de informações , é suficientemente flexível para atender a qualquer aplicação comercial desde que não de cunho essencialmente matemático-estatístico.

MUMPS parece ser a ferramenta ideal para as inúmeras pessoas (pequenas empresas , grupos de pesquisa etc.) que , embora necessitadas de capacidade computacional ON-LINE no seu trabalho do dia a dia , não a obtém devido ao seu elevado custo , sua demasiada complexidade ou simplesmente a dificuldade de acesso aos recursos centralizados . Para estes , uma poderosa "máquina MUMPS" sobre a mesa , de fácil manejo e programação , pode ser a solução ! Para aplicações de maior porte , um mini-computador dedicado ao MUMPS , permitirá a realização de excelentes trabalhos a curto prazo .

É a nossa opinião que as vantagens da utilização de um computador próprio , citadas neste trabalho , poderão ser plenamente aproveitadas utilizando-se o sistema MUMPS.

No barco dos sistemas, o programador será navegador, como sugere Bachman, ou capitão , como defende o MUMPS ? Serão os programadores e ou analistas capazes de elevarem o nível de confiabilidade e manutentabilidade de seus sistemas,ou isto terá que ser atingido por medidas repressivas por parte do S.O., SGBD e a gerência ?

BIBLIOGRAFIA

Adams , C. W. , "Grosch's Law Repealed" , DATAMATION , May 1692

Auerbach , "Problems in Decentralized Computing" , System Developmente Considerations

Auerbach , "Managerial and Economical Issues in Distributed Computing" , General Management , Management Planning

Bachman , C.W. , "The programmer as Navigator" , Communications of the ACM , 16 , 11 , 1973

Brooks , F. P. , Jr. , The Mythical Man-Month , Massachussets , Addisson-Wesley , 1975

Date , C. J. , An Introduction to Database Systems , Massachussets , Adisson-Wesley , 1975

Munnecke , T. H. , "Data Base Managemente Systems - Friend or Foe ? " , Proceedings of the MUMPS Users Group Meeting , Loma Linda , 1975

. . . , "A tiny PASCAL Compiler" , BYTE , Setembro , 1978

O'Neil , J. T. , Editor , MUMPS Language Standard , NBS Handbook 118 , Washington , 1976

Wagner , F. V. ,"Is decentralization inevitable?" , DATAMATION , Nov 1976

Walters , R. F. , Mumps Primer , revised , MUMPS Development Commitee , Bedford , 1979

Wiederhold , G. , Database design , Stanford , McGraw-Hill , 1977

Zimmerman , J. , Editor , "What is MUMPS" , MUMPS Users Group , St. Louis , 1975

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

UNA METODOLOGIA DE PROJETO DE BANCOS DE DADOS PARA UN SOFTWARE ESPECIFICO

V. W. Setzer

R. Lapyda

Instituto de Matemática e Estatística
Universidade de Sao Paulo, Brasil

1. INTRODUÇÃO

Um dos grandes problemas no projeto de bancos de dados usando sistemas comerciais, também chamados "pacotes", é o perigo de se limitar a visão da realidade, reduzindo-a a certas características dos modelos seguidos pelos sistemas em questão. Por exemplo, o uso de pacotes que empregam o modelo hierárquico, como o IMS /DAT 77/ e o SYS-2000 /MRI 72/, pode condicionar os usuários a encararem todas as estruturas de seu problema como hierárquicas. Ou, na melhor das hipóteses, o usuário tem consciência dessa limitação e deve introduzir no projeto de seu banco de dados uma série de artifícios, tais como arquivos auxiliares ou redundâncias, de maneira que o modelo final adapte-se às restrições ou à eficiência do pacote. O projetista pode tender a ficar muito preocupado com esses artifícios, criando um modelo que deixa de ter uma clara correspondência com a realidade. Uma consequência desse fato pode consistir na dificuldade de introduzir alterações no modelo.

Para contornar esses inconvenientes, consideramos interessante o uso de um *modelo conceitual* que não sofre restrições devidas a problemas de implementação. Ele deve ser uma abstração do mundo real em um nível puramente formal, porém com características adequadas a sistemas de informação. Nesse sentido, escolhemos o Modelo Entidade-Relacionamento, "Entity-Relationship Model" (MER), de Chen /CHE 76/ como modelo conceitual. Não entraremos em detalhes desse modelo, fazendo apenas uma breve introdução ao Diagrama Entidade-Relacionamento (DER), que constitui o cerne do MER. Serão abordadas apenas as principais características do DER. A seguir, fazemos breve introdução ao modelo de dados do sistema ADABAS /SOF 71/, inclusive com algumas considerações sobre as suas linguagens de consulta. Finalmente, apresentamos uma metodologia para conversão do DER para os arquivos ADABAS.

Na medida das possibilidades e do espaço, empregaremos uma descrição informal.

2. O MODELO ENTIDADE-RELACIONAMENTO

O modelo introduzido por Chen considera que o mundo real pode ser representado conceitualmente como uma coleção de *conjuntos de entidades (CE)* e de *conjuntos de relacionamentos (CR)* existentes entre entidades. Não se pode definir o que vem a ser uma entidade; a grosso modo, trata-se da abstração de qualquer objeto , ser, ou evento da realidade. Os relacionamentos são associações formais entre elementos de CEs. Por exemplo, podemos ter os CEs ALUNOS e DISCIPLINAS, e um CR HISTÓRICO ESCOLAR. Cada elemento deste associa um elemento do conjunto ALUNOS a um elemento do conjunto DISCIPLINAS. O DER correspondente a esses tres conjuntos seria o da fig.1.

fig.1

Nesse diagrama há uma notação adicional, que é lida m:n ("m para n"), e que caracteriza o tipo do CR HISTÓRICO ESCOLAR. Nesse caso, fica explícito que um elemento de ALUNOS, isto é, um aluno, pode ser relacionado ou associado a mais do que um elemento de DISCIPLINAS e vice-versa. Isto é, interpretando o CR como representando a informação de quais alunos concluíram quais disciplinas, podemos deduzir do tipo m:n que cada aluno pode ter concluído um número qualquer (não limitado) de disciplinas, e que cada disciplina pode ter sido concluída por um número qualquer de alunos.

Vamos estender a notação de Chen, introduzindo números naturais quaisquer em lugar de m e/ou n, e que pode melhor ser explicada por meio do exemplo da fig.2.

fig.2

Podemos associar ao DER da fig.2 a interpretação de que cada aluno pode estar matriculado em até 8 disciplinas, cada disciplina pode ter um número qualquer de alunos nela matriculado.

Em particular, pode-se ter CRs dos tipos 1:n e 1:1. Note-se que os CRs MATRÍCULA e HISTÓRICO ESCOLAR podem coexistir, isto é, dois CEs podem ser relacionados por mais de um CR.

A cada CE ou CR pode-se associar funções (no sentido matemático) que levam cada elemento desses conjuntos a um só elemento de um *conjunto de valores*. Essas funções são chamadas de *atributos*; sua representação diagramática é dada na fig.3, conforme Chen /CHE 77/.

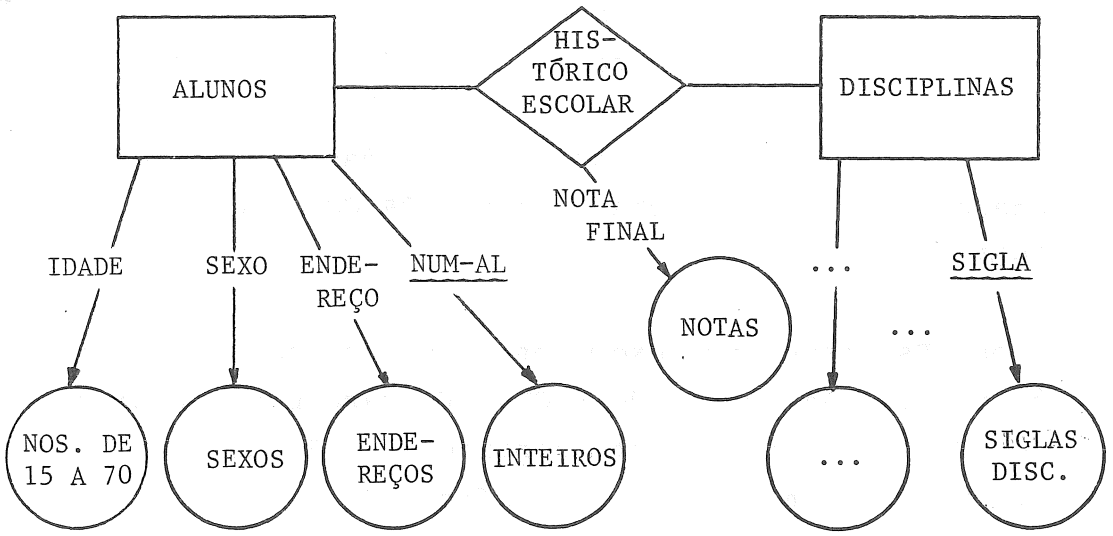


fig.3

Na fig.3, são atributos IDADE, SEXO, ENDEREÇO, NUM-AL, NOTA FINAL; são conjuntos de valores NOS.DE 15 A 70, SEXOS, ENDEREÇOS, INTEIROS, NOTAS, SIGLAS DISC. Sua interpretação pode ser entendida como representando propriedades de cada entidade ou relacionamento. Assim, IDADE é uma função que leva um elemento do conjunto de alunos a um único valor que é um número natural, atribuindo dessa maneira um determinado valor de idade para cada aluno. Da mesma forma, podemos entender NOTA FINAL como uma nota associada a cada par aluno-disciplina, representando com qual nota o aluno concluiu uma determinada disciplina. Note-se que esse atributo origina-se no CR, e não é característica (atributo) de elementos de ALUNOS isoladamente, e nem de elementos de DISCIPLINAS.

Na fig.3 os atributos NUM-AL e SIGLA são especiais, pois constituem-se em *chaves primárias*, ou simplesmente *chaves* dos CEs a que são aplicados. Uma chave é um atributo cuja função é uni-unívoca (isto é, 1 para 1). Assim, para cada elemento do CE a chave leva a um único elemento do conjunto de valores e a esse elemento só corresponde um elemento do CE segundo essa chave. Por exemplo, cada aluno tem apenas um número de aluno e vice-versa. Os valores das chaves servem, assim, para identificar os elementos do CE correspondente. Pode acontecer que uma

chave tenha que ser formada por mais do que um atributo. As chaves serão indicadas sublinhando-se os nomes dos atributos.

Os CRs também tem chaves; estas são constituídas das chaves dos CEs relacionados. Na fig.3, a chave de HISTÓRICO ESCOLAR é o par (NUM-AL, SIGLA).

Note-se que os CRs vistos nos exemplos acima são de classe *binária*, isto é, relacionam apenas dois CEs. Um caso particular de CR binário é o auto-relacionamento binário, um CR que relaciona um CE consigo mesmo, como o exemplo da fig.4.

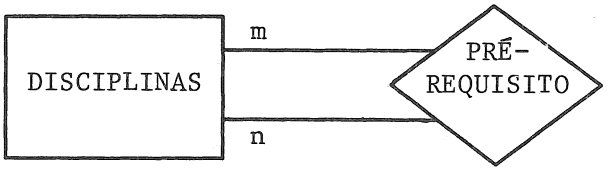


fig.4

Pode-se ter classes de CR relacionando mais do que dois CEs, como por exemplo a classe *ternária*.

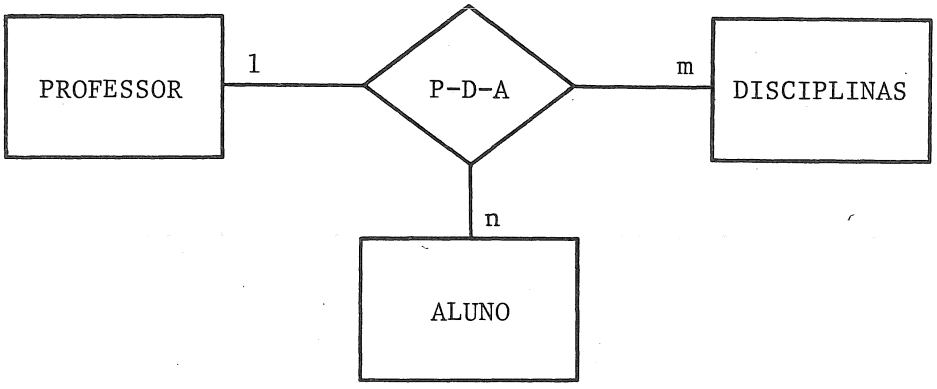


fig.5

Na fig.5 mostramos um CR ternário do tipo 1:m:n. Interpretaremos esse CR como sendo o fato de que a cada par aluno-disciplina corresponde no máximo um professor, isto é, cada aluno só pode ter um professor por disciplina. Por outro lado, um aluno pode estar cursando com um certo professor um número qualquer de disciplinas, e um professor pode ministrar uma certa disciplina a qualquer número de alunos.

Os CRs ternários podem ainda ser do tipo 1:1:n, m:n:p, etc.

Existem interessantes extensões do MER (/S-S 79/, /S-N 79/) que não serão utilizadas neste trabalho, pois é nossa intenção permanecer no mais simples nível, simplificando tanto à compreensão do método a ser exposto, quanto a sua execução em casos práticos. Também não abordaremos casos extremamente particulares, como relacionamentos do tipo exemplificado na fig.6.

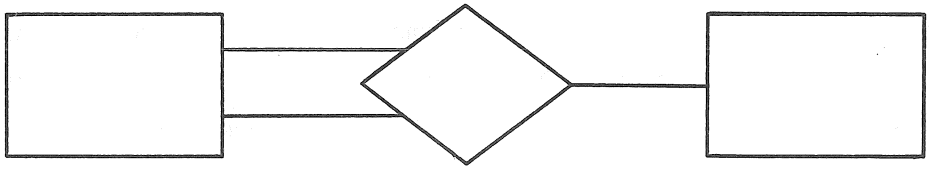


fig.6

3. O MODELO DE DADOS DO SISTEMA ADABAS

Neste item, faremos uma introdução ao sistema ADABAS, principalmente no que concerne ao seu modelo lógico de dados, isto é, como visto pelos usuários. Essa introdução será feita a partir dos conceitos vistos no MER, não se restringindo aos termos empregados normalmente em ADABAS.

3.1. E-ARQUIVOS

Aos CEs do MER faremos corresponder "files" ou arquivos, do ADABAS, e que denominaremos de "entity files" ou E-arquivos. Aos atributos dos CEs corresponderão "fields" ou E-atributos. Os conjuntos de valores do MER não podem ser totalmente especificados, sendo parcialmente definidos pelas características "standard length" e "standard format", e que denominaremos de *formato* do atributo. Por exemplo, as características do CE ALUNOS podem ser representadas em ADABAS por meio da declaração (alguns detalhes serão omitidos):

```
ALUNOS
    01, NA, 6, F, DE     (número do aluno)
    01, ID, 2, F         (idade)
```

```
01, EN, 25, A        (endereço)
01, SX, 1, A         (sexo)
```

Cada linha depois do nome ALUNOS do E-arquivo corresponde a um E-atributo. Nestes, temos especificados, pela ordem: um nível do atributo, como em COBOL ou PL/I; o identificador do atributo; o formato, composto das especificações de comprimento e tipo(F corresponde a inteiro, A a alfanumérico); o primeiro atributo contém uma especificação 'DE', que o caracteriza como um *descritor*, passando com isso a ser um atributo que pode ser usado nas linguagens de consulta dentro dos *critérios de seleção* de registros.

Note-se que nem sempre há uma correspondência total entre os conjuntos de valores do MER e os formatos dos atributos do ADABAS. De fato, no caso do atributo ID, não é possível especificar os números naturais de 15 a 70 como constituindo o conjunto de valores desejado; da mesma maneira não é possível restringir o formato de SX somente aos valores 'M' e 'F'.

Em alguns CEs e em todos os CRs do MER, ocorrem chaves compostas de mais de um atributo. Existe uma correspondência direta destas chaves compostas em ADABAS: são os denominados "superdescriptors", que envolvem mais do que um atributo de um arquivo e funcionam, do ponto de vista dos critérios de seleção, como um único descritor. No restante deste trabalho, consideraremos descritores em geral, sem especificar se são "superdescriptors" ou não.

3.2. CONEXÕES

Os CRs do MER serão representados por construções do ADABAS que denominaremos de *conexões*, que podem ser de duas classes: *implícitas* ou *explícitas*.

3.2.1. CONEXÕES IMPLÍCITAS

Essas conexões são implementações de relacionamentos do MER que usam apenas "couplings" ou *acoplamentos*, entre E-arquivos do ADABAS. Esses acoplamentos tem as seguintes características:

a) Acoplamentos podem ser feitos entre dois e somente dois arquivos distintos, isto é, são sempre binários;

b) Dois arquivos só podem ser acoplados por meio de um único acoplamento. Cada arquivo pode ser acoplado a no máximo 80 outros arquivos;

c) Se os arquivos A_1 e A_2 estão acoplados, existe uma associação implícita entre registros de A_1 e registros de A_2;

d) O acoplamento entre dois arquivos A_1 e A_2 é feito chamando-se dinamicamente uma rotina utilitária do sistema, à qual são fornecidos os identificadores de A_1 e A_2, bem como os identificadores de um atributo de cada um, cuja declaração tenha a especificação 'DE';

e) Se um acoplamento entre dois arquivos A_1 e A_2 é feito através dos atributos t_1 de A_1 e t_2 de A_2, uma associação implícita é estabelecida entre cada registro de A_1 e os registros de A_2 para os quais t_1 e t_2 tem o mesmo valor, e vice-versa. A intersecção dos conjuntos de valores de t_1 e t_2 não deve, portanto, ser vazia;

f) Em um arquivo A, um atributo t declarado simplesmente como 'DE' permite que cada registro de A tenha apenas um valor para t. Se, além de 'DE', t for declarado como 'MU' ("multiple"), cada registro de A poderá conter de zero a 191 valores diferentes de t. Esta característica do ADABAS será empregada na implementação simplificada de certos CRs do MER, como veremos no item 4.

Nas figs. 7 e 8 damos exemplos de conexões implícitas, sem atributos especificados como sendo do tipo 'MU' e, na fig.9, com essa especificação.

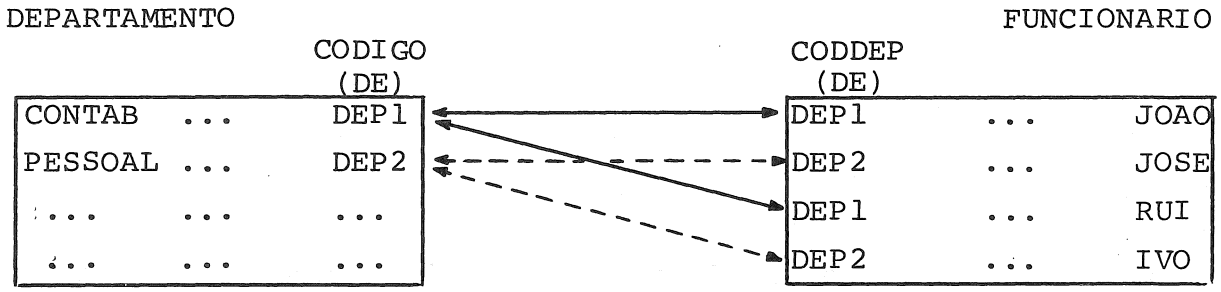


fig.7

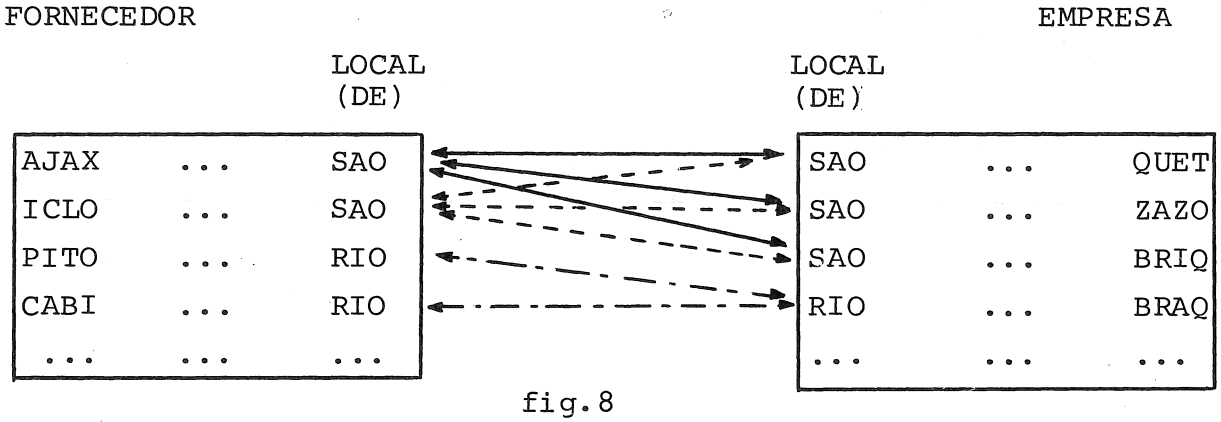


fig.8

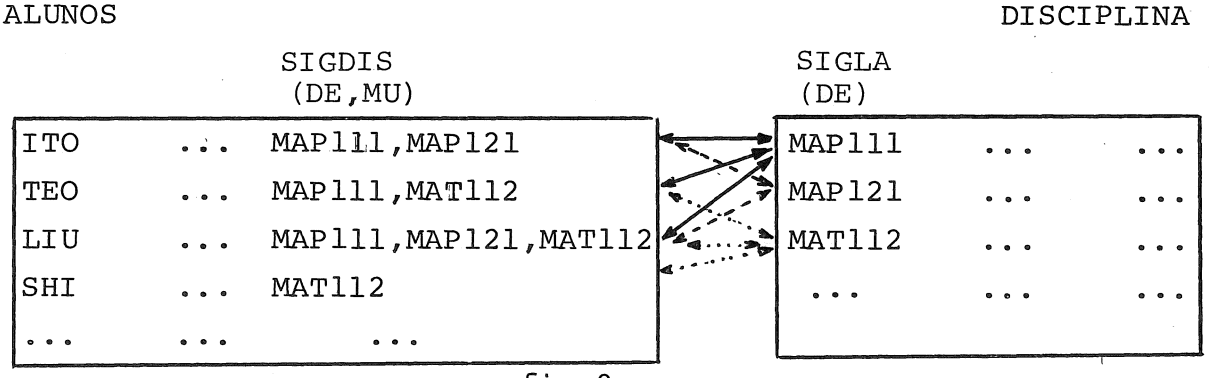


fig.9

Note-se que na fig.7 o atributo chave do arquivo DEPARTAMENTO foi duplicado em FUNCIONARIO para obter-se o acoplamento; o mesmo se deu na fig.9 com o atributo SIGLA de DISCIPLINA que foi duplicado como o atributo 'MU' SIGDIS de ALUNOS. Já na fig. 8, os atributos de nome LOCAL são próprios dos arquivos FORNECEDOR e EMPRESA, não tendo havido necessidade de duplicação.

As conexões implícitas estão sujeitas às restrições dos acoplamentos, como por exemplo o fato de serem sempre binários.

3.2.2. CONEXÕES EXPLÍCITAS

Essas conexões são implementações de relacionamentos do MER pela introdução de arquivos auxiliares que denotaremos por *R-arquivos* cujos registros são formados por *R-atributos*. Para estabelecer-se uma conexão explícita entre dois arquivos E_1 e E_2 (que em geral serão E-arquivos, com uma exceção), é introduzido um R-arquivo R e dois acoplamentos, um entre R e E_1 e o outro entre R e E_2. Para implementar esses acoplamentos, um atributo t_1 de E_1 e um atributo t_2 de E_2 são duplicados em R, constituindo os atributos r_1 e r_2. Uma associação entre um registro de E_1 com um valor v_1 de t_1 e um registro de E_2 com um valor v_2 de t_2 é estabelecida pela introdução de um registro de R com valores de r_1 e r_2 iguais a v_1 e v_2 respectivamente. Na fig.10 damos um exemplo dessa conexão, onde E_1 corresponde a ALUNOS, E_2 a DISCIPLINAS, R a HISTÓRICO ESCOLAR, t_1 a NUM, r_1 a SIGLA, r_2 a SIGD.

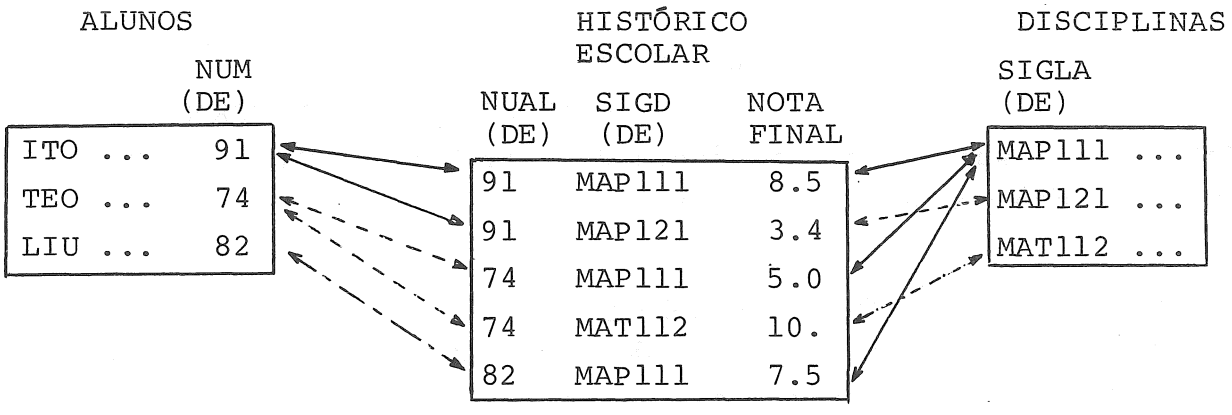


fig.10

Note-se que as conexões explícitas podem ser ternárias, ligando tres E-arquivos; deve-se duplicar um descritor de cada um destes no R-arquivo da conexão, e introduzir-se tres acoplamentos do R-arquivo para cada um dos E-arquivos. Em geral, podemos ter conexões explícitas n-árias.

3.3 LINGUAGENS DE CONSULTAS

Não entraremos aqui nos detalhes das várias linguagens de consulta do ADABAS. Importa-nos descrever uma característica fundamental da linguagem básica de consulta que usa linguagens hospedeiras ("host languages"), bem como da linguagem de consulta direta ("query language") ADASCRIPT e da linguagem independente NATURAL devido ao impacto na simplicidade de formulação e na eficiência das consultas. Trata-se do fato de que as consultas que utilizam acoplamentos só permitirem a seleção de registros de um arquivo A, sujeitos a determinadas condições sobre atributos de A do tipo 'DE', e sujeitos a condições sobre atributos de arquivos A_1, A_2, ...,A_n acoplados cada um *diretamente* a A; essas condições devem formar uma expressão conjuntiva (isto é, do tipo "e") em relação aos vários arquivos. Exemplificando, usando ADASCRIPT,podemos ter, para os arquivos da fig.10:

```
find all records in file HISTORICO-ESCOLAR
        with NOTA-FINAL < 5.0  and
        coupled to file ALUNOS with IDADE > 30
                              and SEXO = 'M'
    and coupled to file DISCIPLINAS
                  with PERIODO = 'NOTURNO'
```

Note-se que supomos a existência dos atributos IDADE e SEXO em ALUNOS e PERIODO em DISCIPLINA.

Antes da palavra reservada coupled só pode ser usada a conjunção and. Assem, tanto não se pode usar outros conectivos lógicos entre acoplamentos, como não se pode *encadear* acoplamentos em uma só consulta, como por exemplo "ache registros de A acoplados a registros de B, que por sua vez estão acoplados a registros de C". Mais concretamente, seria possível formular, para os arquivos da fig.10, a consulta "obter os registros de alunos que tiveram nota final menor do que 5.0 em disciplinas do período noturno" (novamente consideramos que DISCIPLINAS contém um atributo PERÍODO).

4. CONVERSÃO DO MER PARA O ADABAS

Neste item, introduziremos um procedimento para conversão sistemática do DER para o ADABAS. Devemos, para isso, adotar um critério geral, que será o seguinte: os esquemas de conversão serão tais, que permitam buscas na linguagem básica (cujas restrições se refletem nas outras linguagens) usando, sempre que possível, somente acoplamentos, isto é, sem que seja necessário colecionar valores de algum atributo de um arquivo para posteriormente usá-los como valores de descritores na busca de registros do mesmo ou de outro arquivo. Em termos do ADABAS isto equivale a trabalhar, quando possível, apenas com as ISNs que são, essencialmente, endereços dos registros.

Serão construídos dois conjuntos: Q com arquivos e C com acoplamentos. O conjunto Q consistirá de E-, R- e X-arquivos; estes últimos serão definidos mais adiante. Suponhamos que o DER a ser convertido tenha os CEs T_i, i=1,2,...,t e os CRs L_i, i=1,2,...,l. Os passos desse procedimento são os seguintes:

a) *Inicialização:* faça Q:=Φ e C:=Φ onde Φ é o conjunto vazio.

b) *Geração:* nesta fase são criados arquivos que implementam os CEs e os CRs do MER no ADABAS

b_1) Para cada T_i do MER, i=1,2,...,t

- crie um E-arquivo E_i, com os E-atributos correspondentes aos atributos de T_i; a chave de T_i deve ser especificada em E_i como descritor ('DE');

- faça Q:=Q$\cup\{E_i\}$, isto é, introduza E_i em Q.

b_2) Para cada L_i do MER, i=1, 2,...,l

b_{21}) se L_i não for um auto-relacionamento,

- crie um R-arquivo R_i, estabelecendo uma conexão explícita com os E-arquivos que representam os CEs relacionados por L_i, através dos acoplamentos $c_1^i, c_2^i, \ldots, c_p^i$, supondo-se um CR p-ário. Se existirem atributos próprios de L_i, estes são introduzidos em R_i. Os atributos dos E-arquivos que são duplicados em R_i são aqueles que representam as chaves dos CEs. Portanto, se uma destas chaves envolver mais do que um atributo, é necessário de-

clará-la como "superdescriptor" (V. 3.1).

- faça $Q:=Q \cup \{R_i\}$, $C:=C \cup \{c_1^i, c_2^i, \ldots, c_p^i\}$

b_{22}) se L_i for um auto-relacionamento binário, auto-relacionando um CE T_j

- seja E_j o E-arquivo correspondente a T_j. Crie um arquivo auxiliar X_i, cujo tipo denominaremos de *X-arquivo*, tendo um só atributo x_i proveniente da duplicação do atributo e_j de E_j correspondente à chave de T_j. Os valores assumidos por x_i serão os valores de e_j. Crie um acoplamento c_o^i entre E_j e X_i;

- crie um R-arquivo R_i estabelecendo uma conexão explícita entre E_j e X_i através dos acoplamentos c_1^i e c_2^i; os atributos de L_i são introduzidos em R_i;

- faça $Q:=Q \cup \{X_i, R_i\}$, $C:=C \cup \{c_o^i, c_1^i, c_2^i\}$.

c) *Redução:* nesta fase os conjuntos Q e C são diminuídos, pela substituição de algumas conexões explícitas por conexões implícitas. Lembramos que a cada CR L_k corresponde um R-arquivo R_k, k=1, 2,...,l.

c_1) Para cada par de E-arquivos $E_i, E_j \in Q$, para os quais exista *apenas uma* conexão explícita binária através de R_k e dos acoplamentos c_1^k e c_2^k que usam os atributos t_i de E_i e t_j de E_j, respectivamente,

c_{11}) se L_k é do tipo 1:1

- escolha um dos E-arquivos, por exemplo E_i, segundo um ou mais dos crtiérios descritos abaixo, e duplique nele t_j de E_j, dando origem a t_i^j; introduza em E_i os atributos de R_k provenientes dos atributos de L_k;

- execute o procedimento P (descrito após c_{14}).

Critérios de escolha de E_i:

- *Critério de ausência no CR:* Seja n_p o número médio estimado de registros de E_p que não participam da conexão para p=i,j. Escolhe se E_i se $n_i < n_j$.

Aplicando-se esse critério os novos atributos de E_i terão menos valores vazios do que se a duplicação fosse feita no outro sentido. Em particular, esse critério não é muito importante no ADABAS devido às possibilidades de compressão dos registros, que elimina destes os atributos com valores vazios.

- *Critério da frequência de consultas:* Para n consultas ao banco de dados, estima-se em média n_i (n_j) consultas que localizam registros de E_i (E_j) usando-se, entre outros atributos, condições sobre a chave de E_j (E_i) e/ou atributos do CR. Escolhe-se E_i se $n_i > n_j$.

Com esse critério maximizam-se as buscas que não usam acoplamento.

-*Critério de alteração da chave:* Para n consultas ao banco de dados estimam-se em média n_i (n_j) atualizações das chaves de E_i (E_j). Escolhe-se E_i se $n_i > n_j$.

Com esse critério, minimiza-se o número de consultas que alteram ambos os arquivos simultaneamente. Com isso, não só aumenta-se a eficiência, mas também evitam-se problemas de integridade.

-*Critério de estabilidade de presença no CR:* Para n consultas ao banco de dados, estimam-se em média n_i (n_j) *saídas* (cf. definição abaixo) de registros pertencentes a E_i (E_j) da conexão entre E_i e E_j. Escolhe-se E_i se $n_i < n_j$.

Dados dois arquivos A_1 e A_2 ligados por alguma conexão, dizemos que um registro de A_1 *sai* da conexão se ele, após uma operação de atualização, deixa de estar associado (cf. 3.2) a algum registro de A_2.

Com esse critério, minimiza-se o número de registros que devem ser alterados pela movimentação de registros na conexão. Existe uma movimentação de um registro do arquivo A associado a b_1 de B segundo uma certa conexão quando ele passar a associar-se a b_2 de B segundo essa conexão, deixando a associação com b_1, $b_1 \neq b_2$.

c_{12}) se L_k é do tipo 1:n conforme a fig.11

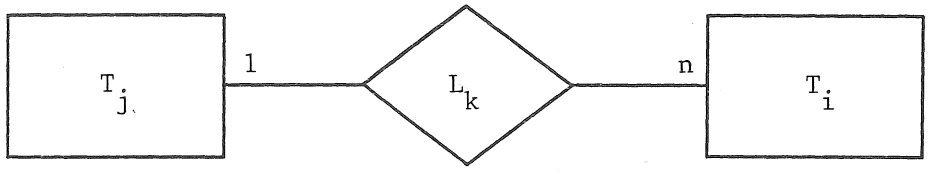


fig.11

- duplique no E-arquivo E_i o atributo t_j de E_j, dando origem a t_i^j; introduza em E_i os atributos de R_k provenientes dos atributos de L_k;

- execute o procedimento P.

Observe-se que há uma possibilidade particular de conversão da conexão explícita em implícita. Trata-se do caso em que L_k é do tipo 1:x, em que $x \leq 191$ (no sentido do item 2), sem atributos. Nesse caso, poderemos optar por duplicar t_i em Ej, dando origem a t_j^i. Este último deve ser declarado com a especificação 'MU' (V. 3.2.1). Existe uma vantagem nessa organização: haverá uma diminuição do tempo de processamento de consultas que solicitam apenas os valores de t_i, isto é, a chave de T_i, usando como critério de seleção valores de atributos de E_j, pois nesse caso o acoplamento não é usado. Por outro lado, existe uma grande desvantagem nessa organização em relação à anterior, sem 'MU': nesta, ocorre um controle automático da integridade, no sentido da preservação do tipo 1:n, enquanto que usando 'MU' essa preservação fica a cargo do usuário.

c_{13}) se L_k é do tipo m:y, em que $1<y \leq 191$, sem atributos, conforme a fig.12

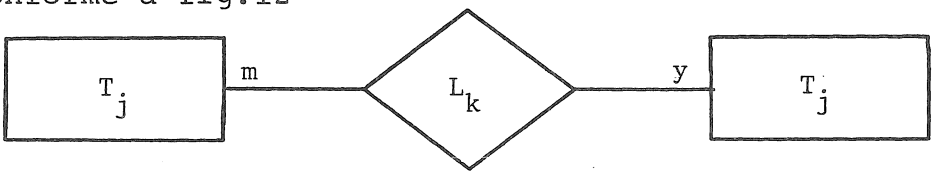


fig.12

-duplique no E-arquivo E_i o atributo t_j de E_j, dando origem a t_i^j com especificação 'MU';

-execute o procedimento P.

c_{14}) se L_k é do tipo x:y em que $1<x,y \leq 191$, sem atributos, conforme a fig.13

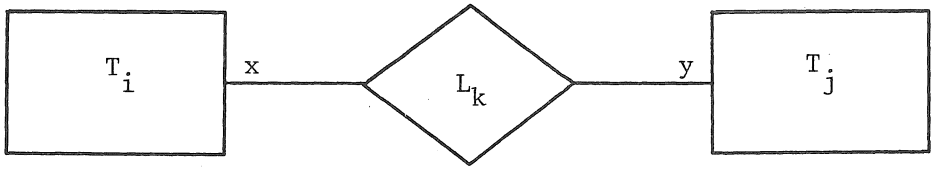


fig.13

-escolha um dos E-arquivos correspondentes a T_i e T_j, por exemplo, E_i, segundo um ou mais dos critérios descritos abaixo, e duplique nele t_j de E_j, dando origem a t_i^j com especifi-

cação 'MU';

-execute o procedimento P.

Critérios de escolha de E_i:

-*Critério de ausência no CR:* Idem ao critério de mesmo nome apresentado em c_{11}.

-*Critério da frequência de consultas:* Idem a c_{11}, lembrando que neste caso não existem atributos do CR.

-*Critério de alteração da chave:* Para m consultas ao banco de dados , estimam-se em média m_i (m_j) atualizações das chaves de E_i (E_j); estima-se que cada registro de E_i (E_j) esteja associado em média a n_i (n_j) registros de E_j (E_i). Escolhe-se E_i se $m_i n_i > m_j n_j$.

-*Critério de movimentação no CR:* Para n consultas ao banco de dados, estimam-se em média n_i (n_j) movimentações (cf. a definição em c_{11}) de registros de E_i (E_j) na conexão correspondente ao CR L_k. Escolhe-se E_i se $n_i > n_j$.

Esse critério tem efeito análogo ao critério de estabilidade de presença de c_{11}.

-*Critério do comprimento dos registros:* Estima-se que cada registro de E_i (E_j) esteja associado em média a n_i (n_j) registros de E_j (E_i); seja s_i (s_j) o comprimento da chave t_i (t_j). Escolhe-se E_i se $n_i s_j < n_j s_i$.

Com isso, conseguem-se registros com menor tamanho, relativamente à parte variável devido ao grau de multiplicidade do atributo especificado com 'MU'.

Procedimento P:

-crie uma conexão implícita entre E_i e E_j, usando t_i^j e t_j, dando origem ao acoplamento c_0^k;

-faça $Q := Q - \{R_k\}$, $C := C \cup \{c_0^k\} - \{c_1^k, c_2^k\}$.

c_2) Para cada par de E-arquivos $E_i, E_j \in Q$, para os quais existam duas ou mais conexões explícitas binárias $R_{k1}, R_{k2}, \ldots, R_{kr}$, $r \geq 2$, correspondendo aos CRs $L_{k1}, L_{k2}, \ldots, L_{kr}$ respectivamente,

c_{21}) se existir um e apenas um CR L_{kp}, $p = 1, 2, \ldots, r$ que satisfaz à restrição c_{1g}, $1 \leq g \leq 4$, execute com esse L_{kp} o procedimento descrito no item c_{1g};

Note-se que se L_{kp} tiver atributos, estes serão incorporados a um E-arquivo; esse fato pode causar problemas para outros CRs, já

que estes não terão nada a ver com os atributos de L_{kp}. Assim sendo, pode-se eventualmente decidir por ignorar o passo c_{21} em alguns desse casos.

c_{22}) se existirem CRs $L_{kp_1}, L_{kp_2}, \ldots, L_{kp_q}$, $1 \leq p_h \leq r$, $q \geq 1$, $1 \leq h \leq q$, satisfazendo cada um a uma das restricões $c_{11}, \ldots, c_{14}$, selecione um L_{kp_h}, segundo um ou mais dos critérios abaixo e execute o procedimento correspondente à restrição satisfeita por L_{kp_h}.

-*Critério de espaço:* Selecione o L_{kp_h} cujo R-arquivo seja estimado como o que ocupa o maior espaço de arquivamento.

-*Critério de eficiência:* Selecione o L_{kp_h} para o qual se estima que a transformação de sua conexão explícita em implícita produza o maior aumento de eficiência às consultas ao sistema.
Esse critério depende não só dos tipos de consulta, mas também da frequência relativa de cada tipo em relação aos outros tipos.

-*Critério da ausência de atributos:* Escolha um dos CRs que tenha o menor número de atributos.
Com esse critério evitam-se ou diminuem-se os efeitos indesejáveis comentados em c_{21}.

5. CONCLUSÕES

Cremos ter mostrado com este trabalho a viabilidade da utilização do MER no projeto de bancos de dados usando pacotes comerciais, tendo para isso empregado o sistema ADABAS com exemplo. É verdade que esse sistema, devido à sua simplicidade, presta-se bem a esquematizações e formalizações. Parece-nos, no entanto, que a mesma metodologia poderia ser desenvolvida para outros pacotes, eventualmente com maior dificuldade.

Por falta de espaço, deixamos de apresentar um diagrama esquemático para o ADABAS, mostrando os vários tipos de conexões entre arquivos, o que poderia ilustrar os passos dos procedimentos de conversão. Pelo mesmo motivo, deixamos de apresentar um exemplo de aplicação.

Um tópico a ser desenvolvido em detalhe seria o de mostrar que o procedimento descrito produz arquivos em ADABAS que, se vistos

como relações do Modelo Relacional /COD 70/, estão em geral em 3ª Forma Normal /COD 72/. Este não é o caso das conexões implícitas que empregam a especificação 'MU', pois aí temos relações que não estão nem em 1ª Forma Normal. Com isso, a metodologia aqui descrita corresponde, praticamente, a um projeto "Top-Down" de relações normalizadas implementadas em ADABAS. A correspondência do ADABAS com o Modelo Relacional foi abordada por /REN 76/ sem que, no entanto, fosse dada uma metodologia de estruturação do modelo.

REFERÊNCIAS

/CHE 76/ Chen, P.P.-S. "The Entity-Relationship Model - Toward a Unified View of Data", *ACM Transactions on Data Base Systems* 1,1, Mar. 1976, pp. 9-36.

/CHE 77/ Chen, P.P.-S. "The Entity-Relationship Model - a basis for the enterprise view of data", Proc. AFIPS 1977 NCC, Vol.46, AFIPS Press, Montvale, pp. 77-84.

/COD 70/ Codd, E.F. "A Relational Model of Data for Large Shared Data Banks", *Comm. ACM* 13, June 1970, pp. 377-387.

/COD 72/ Codd, E.F. "Further Normalization of the Data Base Relational Model", in Rustin, R.(ed.) *Data Base Systems*, Courant Computer Science Symposium 6, Prentice-Hall, Englewood Cliffs, 1972.

/DAT 77/ Date, C.J. *An Introduction to Database Systems* (2nd ed.), Addison Wesley, Reading, 1977.

/MRI 72/ SYS 2000 Publications: General Information Manual, Basic Reference Manual, MRI Systems Corp., Austin, 1972.

/REN 76/ Renaud, D. "ADABAS and the Relational Model", 3rd International ADABAS User's Conference, San Francisco, May 76.

/S-N 79/ Santos, C.S., Neuhold, E.J., Furtado, A.L. "A data type approach to the Entity-Relationship Model", PUC,Rio de Janeiro, 1979.

/SOF 71/ ADABAS Publications: ADABAS Introduction, ADASCRIPT User's Manual, ADABAS Reference Manual, NATURAL - User's Guide, Software AG, Darmstadt.

/S-S 79/ Schiffner,G.,Scheuermann,P. "Multiple views and abstractions with an Extended-Entity-Relatioship Model" Computer Languages 4,1979, pp. 139-154.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

MODELO PARA UN SISTEMA GENERALIZADO PARA RECUPERACION DE INFORMACION

Eduardo Luis Miranda
Ramón Falcón 2137, 7º "A", Buenos Aires

OBJETIVO DEL TRABAJO

Presentar las estructuras de datos, y el modo de operación, de un sistema generalizado para la recuperación de información.

ANTECEDENTES

Una aplicación para la recuperación de información, queda definida, con respecto del usuario, por la información que este puede obtener y por la forma en que puede seleccionar dicha información.

El presente estudio trata acerca de este último aspecto, más el problema de la obtención de estadísticas de la información almacenada (ej: $\bar{x}$, s, mín, máx.).

La selección de información se realiza mediante consultas. Estas especifican un criterio a satisfacer y el resultado de su aplicación contra la base de datos es el conjunto de entidades que satisfacen el criterio fijado.

Podemos definir cuatro tipos de consultas:

C_1 - Consultas simples, donde solamente es especificado un valor por medio del operador relacional "=".
Ej. SEXO = FEMENINO, NUMERO-DE-EMPLEADO = 998.

C_2 - Consultas por rangos, mediante el uso de los operadores re-

lacionales"< ", "<= ", " >", ">= ", se especifica un conjunto de valores. Ej. EDAD <= 50.

C_3 - Consultas funcionales, el valor de referencia es función de la información almacenada. Ej. SALARIO MEDIA (SALARIOS).

C_4 - Consultas boolenas, son combinaciones de los tipos precedentes mediante la utilización de los operadores lógicos "Y" "NO" y "O". Ej.EDAD <= 30 Y PROFESION = ANALISTA DE SISTEMAS

Mediante los reemplazos:

- NUMERO-DE-EMPLEADO por NUMERO-DE-IVA.
- EDAD por AÑOS-EN-PLAZA
- SALARIO por CAPITAL
- PROFESION por RAMA-DE-LA-INDUSTRIA,

nos desplazamos de una aplicación de personal hacia una de información empresaria y los ejemplos mencionados continuan teniendo sentido.

De lo expuesto, bajo C_1, C_2, C_3 y C_4 las consultas a realizarse, sobre una o distintas aplicaciones difieren únicamente en el o los nombres de atributos que caracterizan a las entidades de la aplicación.

Todo lo antedicho se puede extender al problema de las estadisticas.

Por lo tanto desarrollando algoritmos y estructuras, haciendo abstracción de dichos nombres, se puede construir un sistema apto para múltiples aplicaciones.

Este enfoque presenta desde el punto de vista del usuario las siguientes ventajas:

- desaparece la distinción entre consultas programadas y espontáneas.
- se logra para todo tipo de consultas, de una o distintas aplicaciones, una interfase única, lo que simplifica la explotación de información.

respecto del sector de sistemas, las características más destacables son:

- programación única, esto implica un menor costo de mantenimiento.
- uniformidad en los procedimientos de seguridad.
- simplicidad en los procedimientos operativos.

DEFINICIONES PRELIMINARES

Todo elemento almacenado tiene asignado uno o dos números positivos, denominados dirección, que, indican, donde en un espacio de almacenamiento dado, se encuentra registrado. En el caso de que los números sean dos, tienen la forma (P,L), donde P indica página y L Línea.

La página es la unidad de transferencia entre la memoria central y un soporte externo. La línea es la unidad de tratamiento lógico; existiendo k líneas por página.

Directorio es un conjunto de índices, donde cada uno de estos puede ser visto como una colección de pares de la forma (valor, dirección). Si el valor determina univocamente a una entidad,entonces la dirección es la de esa entidad, en caso contrario, la

dirección apunta al primer elemento de una lista de direcciones de las entidades que poseen dicho valor para un atributo dado.

Diccionario es el conjunto de nombres de atributos, descriptores y apuntadores, utilizado en la interpretación almacenada.

Cada elemento del diccionario tiene la forma (atributo, descriptor, dirección) donde la dirección apunta a un índice del directorio, que es el correspondiente a los valores que ha tomado el atributo para las entidades existentes en la base de datos.

Los atributos se agrupan en una de dos categorías:

- Atributos básicos, son aquellos que están presentes para todas las entidades.
- Atributos variables, son los que pueden estar presentes o no dependiendo de la entidad que se considere.

En una aplicación de personal, son ejemplos del primer tipo el APELLIDO y el NUMERO-DE-EMPLEADO en tanto que del segundo lo son NOMBRE-DE-LA-ESPOSA y TITULO-UNIVERSITARIO.

PRINCIPIOS DE OPERACION

Sea una m-upla $(E_1, E_2, \ldots, E_m)$, cuyos elementos llamados entidades, están caracterizados por los atributos a_i, $1 \leq i \leq n$ y $1 \leq j \leq m$, estos atributos pueden asumir uno de dos valores y sea $D^{n \times m}$ una matriz, cuyos elementos $d_{ij} = 1$ si la entidad j posee el atributo i y $d_{ij} = 0$ si no lo posee (ver fig. 1).

$(E_1, E_2, \ldots, E_m) =$

$((a_1, a_2, \ldots, a_n)_1, (a_1, \ldots, a_{k-1}, a_{k+1}, \ldots, a_n)_2, \ldots, (a_2, \ldots, a_{n-1}, a_n)_m)$

	1	2		m
a_1	1	1		0
a_2	1			1
a_{k-1}		1		
a_k		0		
a_{k+1}		1		
a_{n-1}				1
a_n	1	1		1

$D^{n \times m}$

FIG. 1

Para hallar todas las entidades que cumplen con la condición de poseer el atributo k, $1 \leq k \leq n$, es necesario observar la fila k de la matriz D.

Las entidades que satisfacen la condición son aquellas, tales que $d_{ij} = 1$, en ese caso se utiliza j como la dirección de E (ver fig. 2a).

Para resolver consultas boolenas se recuperan las filas necesarias y se opera sobre ellas (ver fig. 2b y 2c).

$$q_k = (1, 0, 0, 0, 1, \ldots, 1, 0) \in D^{n \times m}$$

Las entidades E_1, E_5, ... y E_{m-1} cumplen con la condición de poseer. q_k.

a

$$q_k = (1, 0, 0, 0, 1, \ldots, 1, 0)$$
$$q_g = (1, 1, 0, 0, 0, \ldots, 1, 1)$$
$$q_k \wedge q_g = (1, 0, 0, 0, 0, \ldots, 1, 0)$$

Las entidades E_1 y E_{m-1} cumplen con la condición de poseer. $q_k \wedge q_g$.

b

$$q_k = (1, 0, 0, 0, 1, \ldots, 1, 0)$$
$$q_g = (1, 1, 0, 0, 0, \ldots, 1, 1)$$
$$q_k \vee q_g = (1, 1, 0, 0, 1, \ldots, 1, 1)$$

Las entidades E_1, E_2, E_5, ..., E_{m-1}, E_m cumplen con la condición de poseer. $q_k \vee q_g$.

c

FIG. 1

Para extender el modelo a atributos multivaluados con grado mayor que dos, es necesario modificar la definición de la matriz D. En este caso cada fila corresponde a un par (a_i, V_k), siendo $d_{ij} = 1$ si la entidad j posee el atributo i con valor k.

Este cambio en la definición no afecta la forma de las operaciones.

Las consultas por rango, se resuelven mediante series de "o".

El espacio físico ocupado por D sería desmesurado de no ser por el hecho de que una gran cantidad de d_{ij} son ceros. Recurriendo a las técnicas para el manejo de matrices ralas se soluciona el problema tanto de espacio como de tiempo de exploración.

Una de estas técnicas consiste en particionar D en submatrices fila, almacenando únicamente aquellas que poseen uno o más elementos no nulos. Todas las submatrices de una misma fila dan origen a una lista de direcciónes, salvo en el caso de determinación unívoca (ver definición preliminares).

La cantidad de elementos que contiene cada una de estas submatrices es fijo y está relacionado con el tamaño de página del espacio de datos básicos.

Si a_1 , a_2 , ..., a_k son atributos básicos, ellos son almacenados en el espacio de datos básicos y los a_{k+1} , a_{k+2} , ..., a_n atributos variables son almacenados en el espacio de datos variables (ver fig. 3).

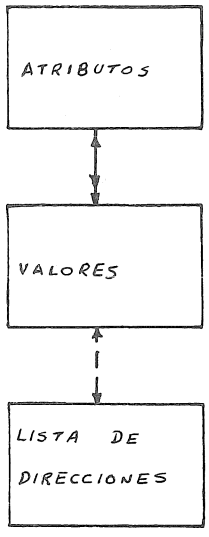
FIG. 3

ESTRUCTURAS DE DATOS

DICCIONARIO / DIRECTORIO

- Funciones: Proveer el registro de los atributos y valores almacenados, de forma tal que permita:
  - Independencia de datos.
  - Localizar rápidamente el conjunto de entidades que satisfacen una consulta dada.

- Estructura: Coexisten en el mismo espacio de almacenamiento tres clases de estructuras (ver fig. 4 y 6):
  - Arbol de atributos
  - Arbol de valores
  - Lista de direcciones



ESQUEMA LOGICO DEL DICCIONARIO/DIRECTORIO

(LA LISTA DE DIRECCIONES, PUEDE EXISTIR O NO DEPENDIENDO DEL ATRIBUTO)

FIG. 4

Características de los árboles de atributos y valores.

Ambos árboles son del tipo B de orden m, y gozan de las siguientes propiedades:

- Todo nodo tiene a lo sumo m hijos
- Todo nodo excepto la raiz y las hojas tienen por lo menos $\lceil m / 2 \rceil$ hijos
- La raíz tiene por lo menos dos hijos (a menos que sea una hoja)
- Todas las hojas aparecen al mismo nivel y no poseen formación (por lo tanto se los puede representar por el apuntador nulo).
- Un nodo que no es hoja, con k hijos, contiene k - 1 claves.

Estructura interna de cada nodo

Arbol de atributos

$(n, D_0, (A_1, D_1, I_1), (A_2, D_2, I_2), \ldots, (A_n, D_n, I_n))$

donde

- $n < m$, cantidad de ternas (A_i, D_i, I_i) en el nodo
- D_i, $0 \leq i \leq n$, son apuntadores a los subárboles del nodo
  A_i, $1 \leq i \leq n$, son los nombres de atributos, y los
  I_i, $1 \leq i \leq n$, son los descriptores de A_i y el puntero al árbol de valores que corresponde al atributo A_i.
- $K_i < K_{i+1}$, $1 \leq i < n$
- Todos los nombres de atributos en el subárbol D_i son mayores (lexicográficamente) que A_i.
- Los subárboles D_i, $0 \leq i \leq n$, cumplen las propiedades anteriores.

Arbol de valores

$(n, D_0, (V_1, D_1, H_1), (V_2, D_2, H_2), \ldots, (V_n, D_n, H_n))$

Se aplican las mismas definiciones que para el árbol de atributo con las siguientes excepciones:
- V_i, es uno de los valores que ha tomado el atributo.
- H_i, es un puntero, que contiene la dirección de una entidad dada, si esta queda univocamente determinada por el par atributo valor, y sino, contiene la dirección del primer elemento de una lista de direcciones. En cualquiera de los dos casos la forma de H_i es (P,L).

Comportamiento de las estructuras (ambos árboles)

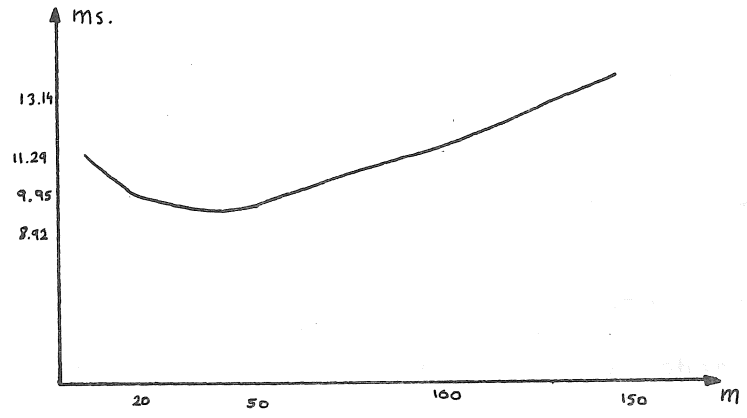
Suponiendo que una vez recuperado el nodo, se realiza una búsqueda binaria sobre las claves, ya sean valores o atributos, el tiempo máximo de búsqueda T para encontrar un tributo o valor es:

$$T \cong (\log_2 (n + 1) / 2) \left(\frac{t_r + t_p}{\log_2 m} + \frac{a \cdot m}{\log_2 m} \right)$$

donde N es la cantidad de claves (atributos o valores de un atributo).

t_r = demora rotacional (promedio)
t_p = tiempo de posicionamiento de cabezas (promedio)
a = $(\alpha + \beta + \gamma)$ tc
t_c = tiempo de transferencia por cada carácter
α, β, γ son la longitud expresadas en caracteres de D_i, A_i o V_i y I_i o H_i según el caso.

Graficando la parte de la expresión que depende de m obtenemos la siguiente curva (ver fig. 5)



$t_r + t_p = 35$ ms

$\alpha + \beta + \gamma = 40$ CARACTERES

$t_c = 10^{-5}$ sec

Se observa que existe un márgen amplio para la elección de m, por lo tanto dentro de ciertos límites, m se fijará teniendo en cuenta las características de la información sobre la cual se va a operar, y la memoria disponible para buffers.

El número máximo de accesos a disco para este tipo de estructuras es:

$$N_a \leq \log_{\lceil m/2 \rceil} (N + 1) / 2 + 1 \qquad (1)$$

Lista de direcciones

Cuando un par de atributo valor no determina univocamente a una entidad, el número de éstas que satisfacen dicho par es desconocido, es necesario entonces adoptar una estructura que provea gran flexibilidad en su capacidad de almacenamiento.

Estructura interna de un elemento de lista.

(D, B, A)

donde:

D, es el número de página de una entidad que satisface el par atributo valor dado. Conceptualmente se lo puede asociar con el subíndice de las submatrices fila mencionadas en Principios de operación.

B, es una cadena de k bits, b_1, b_2, ..., b_k, siendo b_j = 1, $1 \leq j \leq k$, cuando la entidad que satisface el par se encuentra en la línea j de la página D y cero en otro caso. k es igual al número de líneas por páginas existentes en el espacio de datos básicos.

A, es el apuntador al próximo elemento de lista, siendo su forma (P, L).

Para que exista un elemento de lista, correspondiente a la página D, debe existir en esa página por lo menos una entidad que satisfaga el par atributo valor dado.

La lista se mantiene ordenada en forma ascendente respecto de D.

Estrategia de almacenamiento.

Los atributos pueden clasificarse en densos y poco densos.

Son densos cuando sus valores son comunes a un gran número de entidades, este es el caso de atributo SEXO en una aplica-

ción del personal; mientras que el atributo CARGO-QUE-DESEMPEÑA, es poco denso, pues son pocas las personas que desempeñan la misma función.

Para la primera categoría de atributos, es conveniente agrupar todos los elementos de una o más paginas asignadas en forma exclusiva, a los efectos de reducir el número de accesos a un soporte externo.

Para la segunda categoría, esta política se traduce en desaprovechamiento del espacio de almacenamiento; por esto, cuando un atributo valor dado, comparten el espacio de una página con los elementos de otras listas correspondientes también a atributos poco densos.

Número de accesos a disco

Para los atributos densos es:

$$N = r / (l \times f) \qquad (2)$$

donde

- r , es el número de entidades que satisfafe un par dado
- l , es el número de líneas por página del diccionario/directorio
- f , es el número medio de entidades que son referenciadas por un elemento de lista.

ESPACIO DE DATOS BASICOS

Función: Almacenar los atributos básicos

Estructura interna de un vector de atributos básicos

$$(V_1 , V_2 , \ldots , V_n , S , T)$$

donde

- V_i , es el valor asociado con el atributo A_i
- S , es el apuntador al primer elemento de la lista de atributos variables
- T , indica con que imagen ha de interpretarse el primer elemento de la lista de datos variables, esto último será explicado mas adelante.

Estrategia de almacenamiento

Los vectores de atributos básicos se agrupan en páginas, existiendo k de estos vectores (línea) por página.

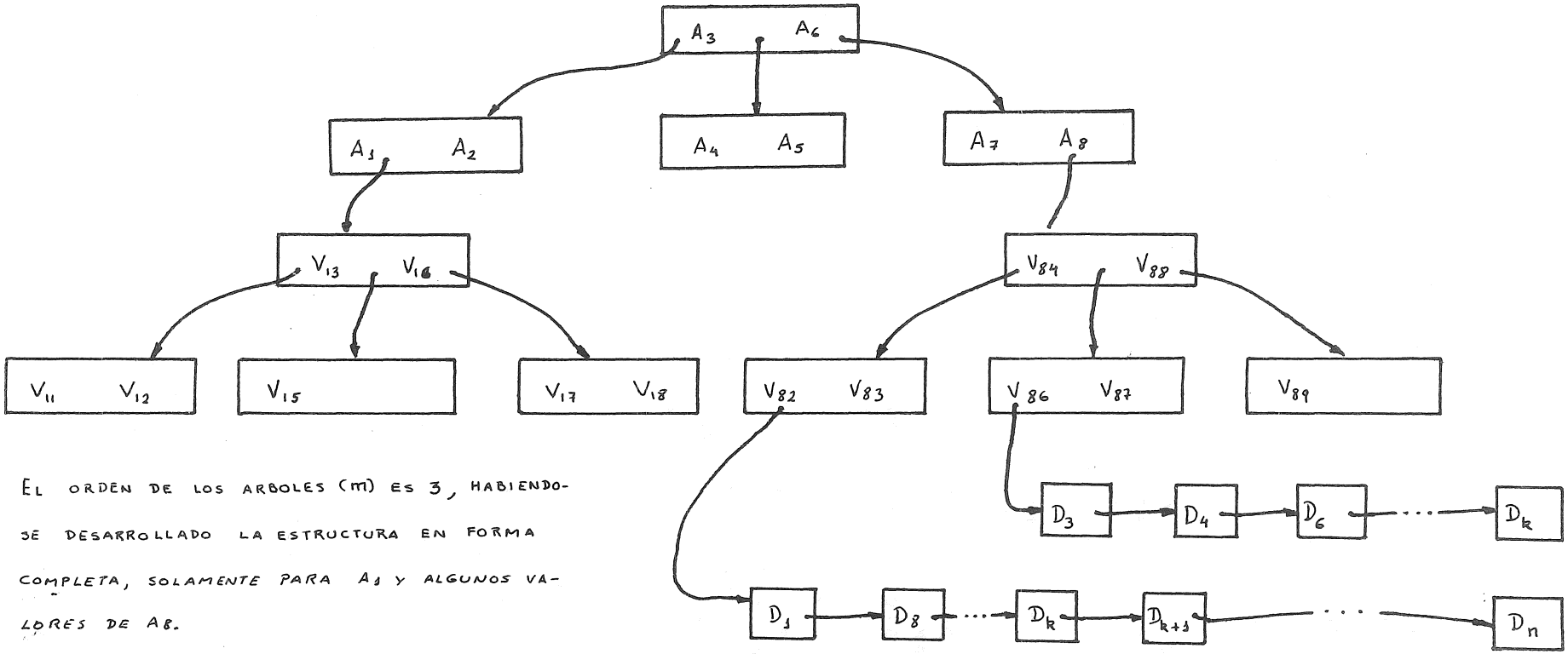
Método de acceso

Los vectores de atributos básicos son accedidos a través de las direcciones (P,L) existentes en el diccionario.

Interpretación de los datos almacenados

Mediante las descripciones existentes en el diccionario

ESTRUCTURA DEL DICCIONARIO



EL ORDEN DE LOS ARBOLES (m) ES 3, HABIENDOSE DESARROLLADO LA ESTRUCTURA EN FORMA COMPLETA, SOLAMENTE PARA A_1 Y ALGUNOS VALORES DE A_8.

A_8, ES UN ATRIBUTO, CUYOS VALORES DETERMINAN UNIVOCAMENTE A LAS ENTIDADES ASOCIADAS, POR LO TANTO, PARA SUS VALORES NO EXISTEN LISTAS DE DIRECCIONES, LO CONTRARIO OCURRE CON A_8.

FIG. 6

DATOS VARIABLES

Función: Almacenar los atributos variables

Estructura: Los atributos variables, se organizan en forma de lista ordenada, respecto del código del atributo.

Estructura interna de un elemento de la lista de atributos variables.

(X, V, S, T)

donde

X, es un código de atributo, mediante él se establece la vinculación entre el valor V y su significado y descripción, almacenados en el diccionario.

S, T, se aplican las mismas definiciones que en el caso de atributos básicos.

Estrategia de almacenamiento.

Los elementos de una lista de atributos variables se agrupan a nivel de página, para minimizar el número de accesos a dispositivos de almacenamiento externo.

Puesto que los valores que toman los distintos atributos pueden requerir distintas capacidades de almacenamiento, el espacio de una página es múltiplemente definido, seleccionándose una definición (imagen) en función de la cantidad de caracteres que necesita un valor para ser almacenado.

En el campo T se indica cual es la imagen a utilizar para recuperar el próximo elemento de la lista.

Cuando la capacidad de una página es excedida, se habilita una nueva página aplicándose a ella las definiciones anteriores.

PROCEDIMIENTOS PARA LA OPERACION DEL SISTEMA

Recuperación de información.

Dada una consulta, se determina mediante el uso del diccionario / directorio, que entidades satisfacen el criterio de recuperación especificado.

Sea por ejemplo una consulta de tipo 1. Para este caso se procede a buscar en el árbol de atributos, aquel por el cual se ha preguntado. Una vez hallado dicho atributo, mediante su información asociada I_i (descriptor y puntero a la raíz del árbol de valores), accedemos al árbol de valores correspondiente. En dicho árbol se busca el valor que satisface la condición especificada.

Se presenta aquí dos casos:

- El par atributo valor determina univocamente a una entidad luego H_i (ver árbol de valores), es la dirección de la entidad buscada.
- El par atributo valor determina un conjunto de entidades.
  En este caso H_i es la dirección del primer elemento de una lista de direcciones. Esta lista es recorrida ele-

mento a elemento, utilizándose el siguiente procedimiento para generar las direcciones de las entidades correspondientes:

- D, es el número de página del espacio de datos básicos, en que una de las entidades que satisface el par, está almacenada.
- Recorriendo luego la cadena de bits B, y para cada $b_i = 1$ se genera la dirección (D, i), donde i es un número de línea dentro de la página D.

Mediante las direcciones obtenidas (no importa el caso), se accede al vector de atributos básicos de la entidad correspondiente y luego a traves del puntero S y de acuerdo con la imagen indicada por T a la lista de atributos variables.

Mediante un proceso de edición se elaboran los datos para presentarlos en forma adecuada y se emite una respuesta. Este proceso se repite hasta agotar la lista de direcciones.

Para distintos tipos de consulta, solo varía la forma en que los árboles son recorridos y la necesidad de incorporar para las consultas de tipo 3 y 4 un procedimiento de cálculo y uno de operaciones lógicas. Una vez desarrollado el procedimiento genérico para cada tipo de consulta se habrá obtenido un procesador de consultas de uso general.

Incorporación de una entidad a la base de datos

Como resultado de este proceso, los atributos de una nueva entidad son incorporados a los espacios de datos y el diccionario / directorio es actualizado.

Se determina en primer lugar, donde, en el espacio de datos básicos ha de ser almacenado el vector correspondiente a la nueva entidad.

Para la actualización del diccionario / directorio, se han de considerar los siguientes casos:

- El par atributo valor considerado determina univocamente a una entidad.
- El par atributo valor, no determina univocamente a una entidad y es además denso.
- El par atributo valor, no determina univocamente a una entidad y es poco denso.

Para cada atributo valor se buscará primero en el árbol, de atributos, el atributo considerado, luego se accederá al arbol de valores buscándose el valor especificado. Dicho valor puede hallarse o no. Para esta última alternativa, el valor es incorporado al árbol colocándose en H_i la dirección de la entidad en el espacio de datos básicos o bien la dirección del primer elemento de la lista de direcciones asociadas, en este caso debemos proceder a crear el primer elemento de la lista.

Si el valor hubiese sido hallado, y el par determinara a la entidad en forma univoca, se presenta una condición de error, si el par no determina a una entidad univocamente, entonces se procede a actualizar la lista de direcciones.

En la actualización de la lista se presentan dos situaciones, se incorpora un nuevo elemento a la lista, o se "enciende" un bit

en la cadena B de un elemento ya existente. Esto ocurre, cuando en la página donde va a almacenar el vector de atributos básicos de la nueva entidad, ya existe otra entidad que satisface el par.

La característica de denso, o poco denso influye únicamente en la forma de seleccionar la página del diccionario / directorio en que ha de ser colocado un nuevo elemento de lista.

En todos los casos se deben respetar las características de los árboles.

Luego se crea el vector de atributos básicos y se almacena en el lugar previamente determinado. De igual forma se procede con la lista de atributos variables, determinando para cada uno de sus elementos la imagen a utilizar en el almacenamiento y encadenando los elementos en forma ordenada respecto del código de atributo.

El proceso se repite para todos los pares atributo valor que caracterizan a la entidad a incorporar.

Incorporación de la definición de un nuevo atributo en una base de datos operativa.

Como resultado de esta operación, la definición de las entidades que componen la base de datos se ve modificada por el agregado de un nuevo atributo.

Esta nueva definición no afecta los algoritmos presentados, pues ellos operan, como se ha visto, sobre las definiciones existentes en el diccionario, alcanzándose de esta forma la independencia de datos.

El procedimiento de incorporación consiste simplemente en la actualización del árbol de atributos, mediante la incorporación del nuevo nombre de atributo y su descripción asociada. En esta incorporación deben respetarse las condiciones de existencia de un árbol tipo B.

Obtención de estadísticas

Para la obtención de estadísticas, se localiza, mediante el árbol de atriburos, el árbol de valores correspondiente, y se lo recorre contabilizando los datos necesarios para la estadística requerida.

ANALISIS DEL COMPORTAMIENTO DEL SISTEMA EN TERMINOS DEL NUMERO DE ACCESOS NECESARIOS PARA RESPONDER A UNA CONSULTA.

El número total de accesos, está dado, por la suma de los accesos al diccionario / directorio y a los espacios de datos.

El número de accesos a los espacios de datos es fácilmente calculable, pues usualmente se requiere un acceso a una página del espacio de datos básicos y uno o más accesos a las páginas del espacio de datos variables para recuperar la lista de atributos variables de la entidad correspondiente.

$$\text{Accesos a datos} = \bar{A}_b + \bar{A}_v \leq 1 + \bar{A}_v \leq 2^{(*)}$$

(*) si el tamaño de página es adecuado.

El número de accesos al diccionario / directorio, se ve afectado por un gran número de parámetros que intervienen en el análisis. Estos son:

- Tipo de consulta
- Orden de los árboles de atributo y de valor
- Número de entidades en la base de datos
- Características del par atributo valor al que se hace referencia
- Cantidad de atributos definidos
- Líneas por página de las listas de direcciones
- Número medio de entidades a las que hace referencia cada elemento de la lista de direcciones

De acuerdo con todo ésto, el estudio se limitará al trazado de curvas (ver fig.9), para valores dados (ver apéndice).

El elevado número de accesos correspondientes a la consulta tipo II, en relación con las tipo I y IV, puede ser corregido aumentando el número medio de entidades a las que hace referencia un elemento de una lista de direcciones. Esto se logra mediante la asignación de direcciones a los vectores de atributos básicos de acuerdo con un criterio de agrupamiento(clustering), basado en aquellos atributos sobre los cuales se van a efectuar consultas por rango.

ESTRUCTURA DE LOS ESPACIOS DE DATOS BASICOS Y VARIABLES

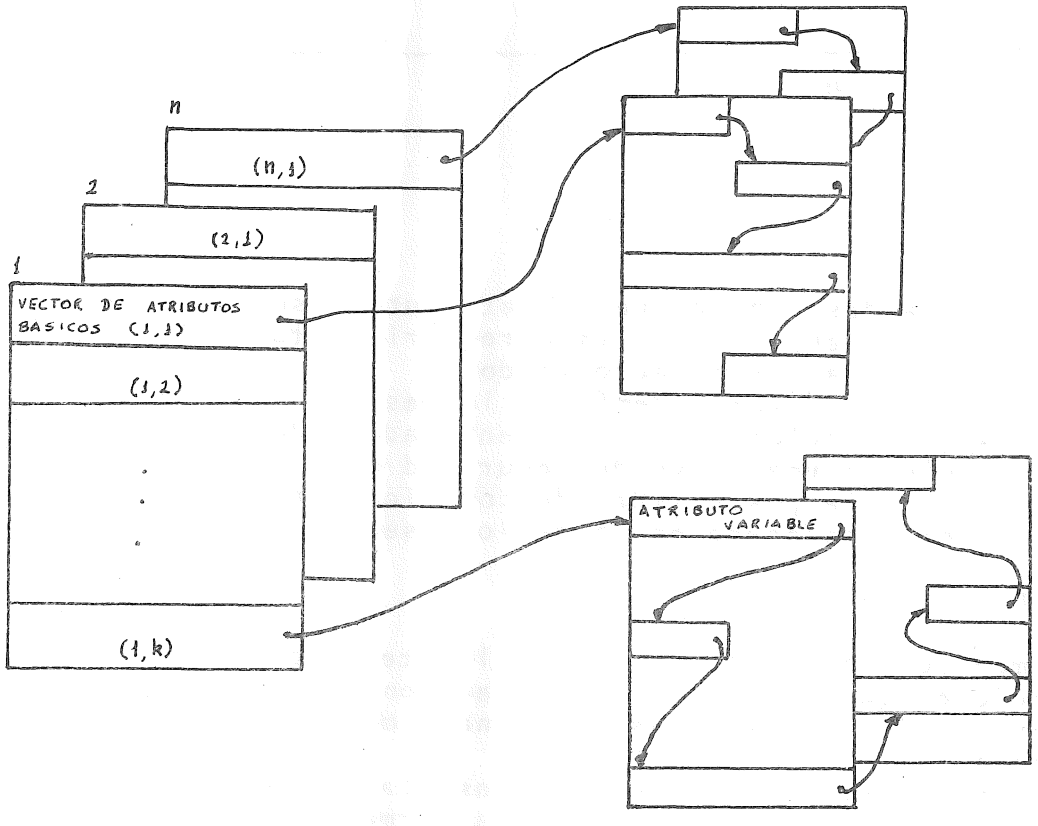


FIG. 8

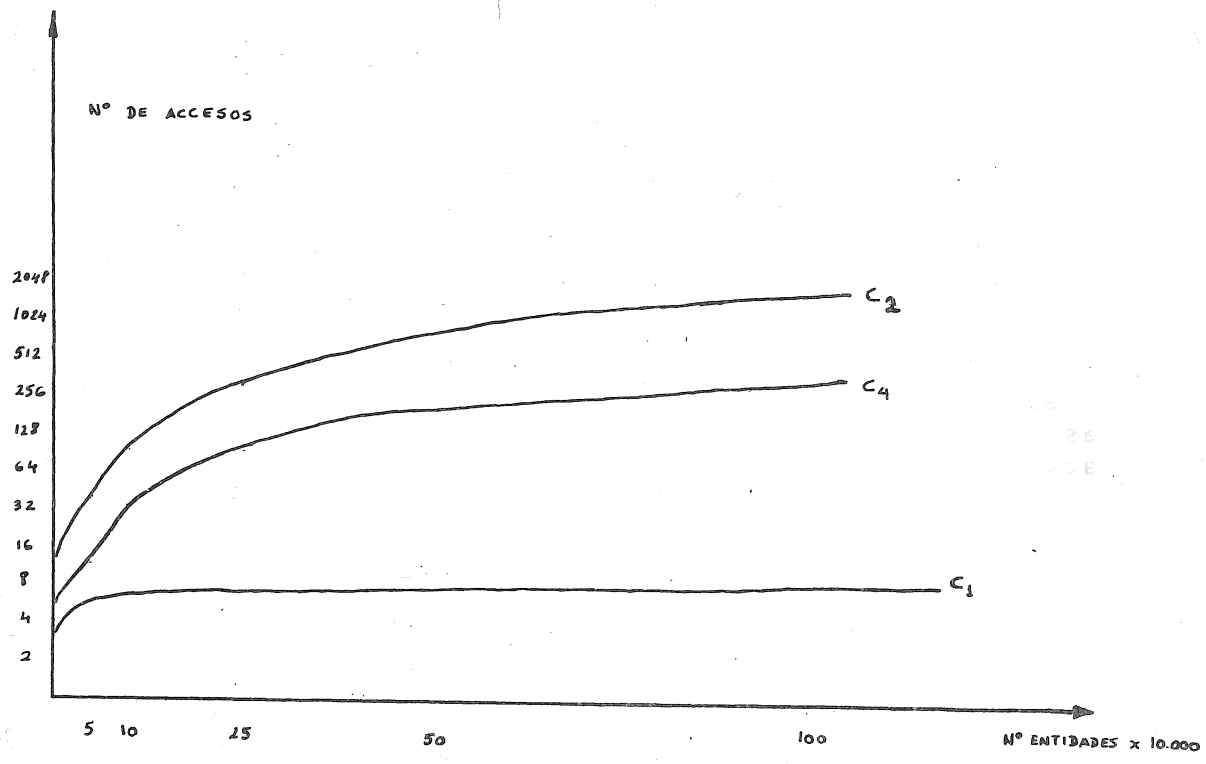


Fig. 9

SEGURIDAD

Junto a la descripción de los atributos, se deben incorporar claves que impidan la consulta o requerimiento de información por parte de usuarios no autorizados.

Se ha de considerar, además de la restricción a nivel de atributos, la restricción a nivel de entidades, debiendo incorporarse a este efecto una clave de seguridad en el vector de atributos básicos; y la posibilidad de codificar la información si esta ha de ser almacenada en un medio inseguro.

AREAS DE APLICACION

El modelo propuesto, es especialmente apto para aquellas organizaciones que necesitan consultar, en forma selectiva extensas bases de datos, y satisfacen una o más de las siguientes condiciones:

- no existen criterios de consulta predominante
- existen varias aplicaciones formalmente iguales
- la definición de las entidades almacenadas, evoluciona constantemente mediante el agregado y / o eliminación de atributos.

ADMINISTRACION DE ESPACIO

La política de administración de espacio, es función de la forma en que se determinan las direcciones de las entidades (vector de atributos básicos) almacenadas en la base de datos

Si no se aplica respecto de las entidades, ningún criterio de agrupamiento, el manejo de espacio puede basarse en listas de espacios recuperados y espacios nunca utilizados.

Esta técnica se implimenta para todas las estructuras del sistema con las siguientes particularidades:

Diccionario:

Una lista para aquellas páginas que contienen al menos una línea disponible y la página en cuestión es utilizada para almacenar los elementos de la lista de direcciones de atributos poco densos.

Dentro de cada página las líneas disponibles son marcadas mediante valores especiales.

Una lista de aquellas páginas que habiendo sido usadas, actualmente no contienen ninguna línea activa.

Una página que contiene una o más líneas (pero no todas) disponibles, y que es utilizada para almacenar los elementos de una lista de direcciones correspondientes a un atributo denso, no pertenece a ninguna lista de espacio recuperado, aunque sus líneas son marcadas como disponibles para ser vueltas a usar por elementos de la lista de direcciones correspondientes al par atributo valor dado.

Este mecanismo asegura el uso de estas páginas en forma exclusiva.

Espacio de datos básicos:

Existe una lista que vincula las páginas que poseen líneas disponibles, dentro de cada página dichas líneas son marcadas.

Espacio de datos variables:

Existe una lista a la cual pertenecen todas aquellas páginas que habiendo sido usadas ya no lo son.

Una página que contiene atributos variables de una entidad, aunque no esté completamente ocupada, no pertenece a ninguna lista, ya que las páginas son de uso exclusivo de una entidad.

Dentro de la página los espacios libres son marcados.

Cuando se aplica un criterio de agrupamiento, el manejo del espacio, es parte principal de la implementación de dicho criterio. Por tal razón no es tratado aquí.

IMPLEMENTACION PARA ATENDER CONSULTAS RESPECTO DE DIFERENTES TIPOS DE ENTIDADES

Lo más indicado para sortear este problema, es introducir un nivel adicional en el diccionario / directorio (ver fig. 10).

Mediante este nuevo nivel, se determinará, el árbol de atributos sobre el cual se ha de operar y luego se procederá en la forma anteriormente descripta.

Otro tipo de solución, menos elegante, que la expuesta consiste en generar múltiples instancias del sistema (ej. catalogar programas y archivos bajo distintos nombres), una instancia para cada aplicación.

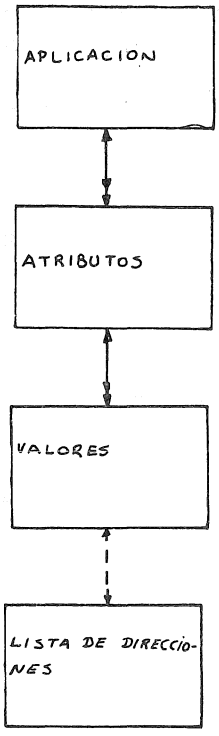


FIG 10

APENDICE

En el cálculo de los valores graficados se han aplicado las expresiones (1) y (2), habiéndose asumido los siguientes valores:

- m: orden de los árboles = 120
- A: Cantidad de atributos definidos = 100
- l: Líneas por página, cuando se almacena un elemento de una lista de direcciones = 200
- k: líneas por página del espacio de datos básicos = 32

En el caso de las consultas II y IV, se aplica la siguiente tabla: (ver fig. 11)

Curva I, consulta simple respecto de un par que determina en forma univoca a una entidad.

R_1 = accesos para localizar el atributo

$\leq \log_{\lceil m/2 \rceil} (A + 1) / 2 + 1$

$\leq \log_{60} 50,5 + 1 \leq 1,96$

R_2 = accesos necesarios para localizar el valor

$\log_{\lceil m/2 \rceil} (N + 1) / 2 + 1$

$\log_{60} (N + 1) / 2 + 1$

$R : R_1 + R_2 = 2,96 + \log_{60} (N + 1) / 2$

Curva II, consulta por rango

$R_1 \leq 1,96$

$R_2 \leq \log_{60} (v + 1) / 2 + 1$

R_3 = número medio de accesos originados en el recorrido de una lista de direcciones $\leq r / (1 * f)$

Respecto de esta consulta, se supone además que se resuelve mediante una serie de operaciones "o", y el valor de referencia es la mediana del conjunto de valores.

$$R \leq R_1 + R_2 + \frac{(v + 1)}{2} R_3 \leq 2,96 + \log_{60} \frac{(v + 1)}{2} + \frac{(v + 1)}{2} \quad \frac{r}{200 f}$$

Curva IV, consulta boolena de la forma $A_1 = V_1$ y $A_2 = V_2$

$R_1 \leq 1,96$

$$R_2 \leq \log_{60} \frac{(v + 1)}{2} + 1$$

$R \leq r / (1 * f)$

Para resolver este tipo de consulta, es menester buscar dos atributos, dos valores y recorrer dos listas de direcciones, por lo tanto:

$$R = 2 \left(1,96 + \log_{60} \frac{(v + 1)}{2} + 1 + r / 200 f\right)$$

$$= 5,92 + 2 \log_{60} \frac{(v + 1)}{2} + r / 100 f$$

Número de entidades en la base de datos	Número de valores que toma el atributo	Número de entidades que poseen el valor	Número medio de entidades que un elemento de lista referencia *
(N)	(v)	(r)	(f)
50.000	8	5.000	3,2
100.000	10	6.250	2,0
250.000	13	12.500	1,60
500.000	15	23.400	1,50
1.000.000	17	41.600	1,33

Fig 11

* Suponiendo una distribución uniforme de las entidades que contienen un valor dado, a través de toda la base de datos,

$$f = (r * 32) / N$$

BIBLIOGRAFIA

- Fundamentals of Data Structures. Horowitz - Sahni. Computer Science Press.
- The Art of Computer Programming , Vol. I y III. Donald Knuth. Addison Wesley.
- Systems Programming. David Hsiao. Addison Wesley.
- Organización de las Bases de Datos. James Martin. Prentice - Hall.
- The Design and Analysis of Computer Algorithms. Aho - Hopocrof - Ullman. Addison Wesley.
- Diseño de Programas para Sistemas. Gauthier - Ponto. Paraninfo.
- Data Structures. Berztiss. Academic Press.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

MTSQL – LENGUAJE DE CONSULTA PARA EL SISTEMA ADMINISTRADOR DE BASES DE DATOS METASYS

A. Morales P.
O. Mascaró G.
Universidad Católica de Valparaíso
Av. Brasil 2950, Valparaíso, Chile

R E S U M E N

El Sistema Administrador de Bases de Datos METASYS constituye una implementación muy cercana al modelo de entidades y relaciones propuesto por P. Chen en 1976, que ha sido adaptado para su aplicación en instalaciones de tamaño reducido. El modelo de entidades-relaciones, sin embargo, puede también ser observado como un conjunto de tablas componentes de un modelo relacional, para efectos de la aplicación de un lenguaje de consulta tipo SEQUEL. Aprovechando las características propias del modelo empleado por METASYS, se ha diseñado un lenguaje de consulta cuyo formato es mas simple que aquel de SEQUEL pues solo debe declarar la tabla objetivo, junto al camino trazado para la consulta y los predicados asociados a la búsqueda. El presente trabajo describe las características de METASYS además de las principales propiedades y limitaciones del lenguaje de consulta MTQL.

1.-INTRODUCCION

En atención al tamaño predominante de las instalaciones computacionales existentes en nuestro medio, el Centro de Computación de la Universidad Católica de Valparaíso decidió en 1976 orientar una parte importante de sus esfuerzos de investigación al desarrollo de herramientas que hiciesen mas eficaz el proceso de construcción de software en instalaciones reducidas.

Como producto de este trabajo, ha sido publicada una serie de artículos donde se describe un conjunto de metodologías que han sido desarrolladas a la luz de las características, actitudes y aptitudes que exhibe el medio mencionado [1 , 2 , 3]. Estas metodologías han sido asimismo divulgadas ampliamente a través de cursos y seminarios, demostrando la validez de las hipótesis de trabajo que las sustentan.

Paralelamente al desarrollo de metodologías de producción de software, se ha sostenido un permanente interés en la automatización de los mecanismos de almacenamiento de información, derivando obviamente hacia el área de bases de datos [4 , 5]. Como producto final de estos proyectos, se ha construído un sistema administrador de bases de datos denominado METASYS [6], inspirado principalmente en el modelo entidades-relaciones propuesto por Chen en 1976 [7].

El sistema administrador de bases de datos METASYS ha servido de base a la implementación de varias aplicaciones que actualmente se encuentran en normal explotación, demostrando en la práctica tener no solo importantes atributos desde el punto de vista de las facilidades que otorga al programador en cuanto al manejo intuitivo del modelo, sino además positivas ventajas relativas a la administración global de los datos de la instalación.

Una vez alcanzados los objetivos asociados a la disponibilidad de una herramienta destinada a la administración de datos (METASYS), ha sido necesario abordar el desarrollo de un mecanismo de consulta que, orientado preferentemente al usuario final, permita interrogar facilmente la base datos. Ello ha dado lugar al lenguaje MTSQL, el cual no es otra cosa que una adaptación de los conceptos empleados por aquellos lenguajes que se sustentan en el álgebra relacional [8] a los requerimientos del modelo METASYS. En las siguientes secciones se describirá con algún detalle las características del modelo METASYS, con el propósito de discutir a continuación las propiedades del lenguaje de consulta MTSQL.

2.-EL MODELO METASYS

Intuitivamente, es fácil advertir la existencia de entidades (concretas o abstractas) durante el proceso de modelación del mundo real cuando se diseña una base de datos. Asimismo, entre las entidades

definidas se deriva, casi naturalmente, un conjunto de relaciones que establecen las asociaciones existentes en el mundo real. Si se toma el ejemplo de un conjunto de datos donde participan alumnos y asignaturas, es posible distinguir la entidad "alumno" y la entidad "asignatura" como elementos independientes dentro del modelo, pero que pueden encadenarse mediante la relación "inscripción", la cual llevará implícita la propiedad de que un alumno tenga varias asignaturas inscritas y, a la vez, que una asignatura tenga varios alumnos inscritos (Fig. 1).

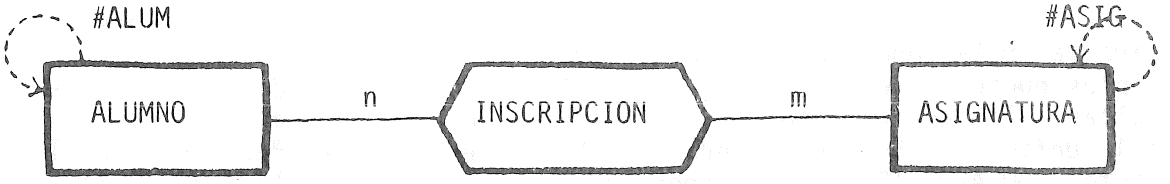


Figura 1.- Esquema básico entidad-relación

El sistema METASYS permite modelar relaciones m:n (BETWEEN-AND) como aquella del ejemplo considerado, asi como relaciones del tipo 1:n (FROM-TO) que también se encuentran frecuentemente en la práctica (p. ej.: la relación asignatura-facultad, donde una facultad ofrece varias asignaturas, pero una asignatura dada solo es ofrecida por una sola facultad en forma exclusiva).

A cada relación es posible atribuir ciertas restricciones que sirven de base para otorgar una mayor consistencia al modelo con respecto del mundo real. En efecto, toda relación puede ser declarada como UNIQUE o NONUNIQUE, esto es, atribuirle a cada posible combinación entre dos elementos determinados de dos diferentes entidades, la propiedad de establecer relaciones con una sola ocurrencia permitida (UNIQUE), o la aceptación de varias ocurrencias (NONUNIQUE). En el ejemplo de alumnos y asignaturas anteriormente presentado, la relación "inscripción" es del tipo UNIQUE por cuanto un alumno solo está autorizado para tener una sola inscripción en una asignatura dada. Sin embargo, si el modelo en diseño está destinado a almacenar información histórica, esto es, datos de varios semestres, la relación "inscripción" deberá ser declarada como NONUNIQUE, puesto que un alumno podrá inscribirse mas de una vez en una misma asignatura en semestres diferentes, debido a la reprobación eventual de la asignatura en algún semestre anterior.

METASYS permite también modelar la presencia de relaciones que no tengan ningún contenido. La relación "inscripción" tiene una serie de atributos propios, tales como; fecha de inscripción, número del documento de inscripción y resultado final obtenido en la asignatura. Sin embargo, una relación como "asignatura-ofrecida", entre las entidades "facultad" y "asignatura", constituye una relación 1:n sin contenido propio, reconocida en METASYS como NULL.

Asímismo, la simple estructura jerárquica (FROM), también es modelable en METASYS, lo cual completa convenientemente las posibilidades del sistema para la representación apropiada de las situaciones encontradas habitualmente en la práctica del diseño de bases de datos. Este es el caso del eventual registro de las características de las distintas "evaluaciones" realizadas para cada asignatura durante el semestre, que constituye una ocurrencia múltiple de registros dependientes de una entidad.

Adicionalmente, METASYS presenta la propiedad de aceptar la especificación de índices secundarios, tanto para las entidades como para las relaciones, otorgando al modelo la capacidad de establecer múltiples puntos de entrada que atribuirán una mayor eficiencia a ciertas consultas a la base de datos. En el ejemplo de alumnos y asignaturas, es posible definir un índice secundario sobre la entidad "alumno" en base al atributo CEDULA-DE-IDENTIDAD, suponiendo que toda persona lo tiene asignado. Este índice será declarado como NODUP, esto es, que no aceptará duplicaciones puesto que se trata de un atributo cuya repetición sería errónea. Asimismo, es posible definir un índice secundario sobre la relación "inscripción", índice que a diferencia del anterior aceptará necesariamente duplicaciones (DUP), puesto que más de una inscripción podrá haberse efectuado en una misma fecha.

Una de las característica releventes del modelo entidades-relaciones, es su contribución en materia de representación gráfica de una base de datos. El sistema METASYS ha tomado como base las proposiciones que en este contexto fórmula Chen[7], adaptándolas a las particularidades de su propio modelo. La figura 2. presenta un esquema asociado al ejemplo que ha servido para ilustrar las características de METASYS en este trabajo. Una paralelógramo identifica a las entidades, mientras un hexágono identifica a las relaciones. El tipo de relación (n:m ó 1:n) queda explícitamente declarado en las líneas que acompañan a la relación. Asimismo, la propiedad de única (UNIQUE), no única (NONUNIQUE) o nula (NULL), queda directamente indicada sobre cada relación, salvo en el caso de la relación jerárquica, la cual va unida a una flecha desde su entidad dueña.

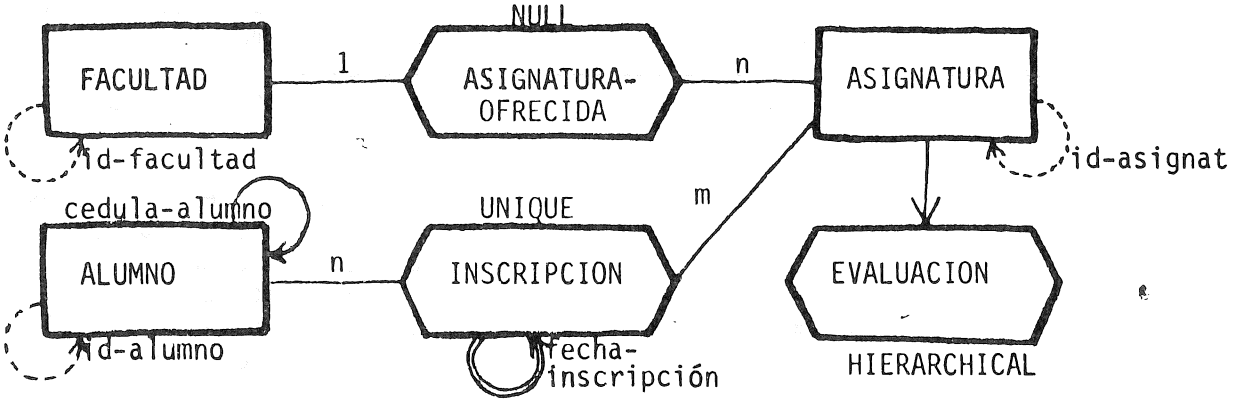


Fig. 2.- Representación gráfica del modelo.

La representación gráfica de los índices, se efectúa mediante semi-círculos dibujados sobre cada objeto indexado. La clave o índice primario, obligado para cada entidad, se declara mediante un semi-círculo con línea punteada. Un índice secundario que no acepta valores duplicados se representa mediante un semi-círculo en línea llena, y aquellas que aceptan valores duplicados, mediante una doble línea llena.

3.- LENGUAJES METASYS-DDL Y METASYS-DML

Toda base de datos modelada en función de las propiedades descritas para METASYS, puede ser expresada mediante el lenguaje de descripción de datos que el sistema dispone para ello (METASYS-DDL). Este lenguaje constituye una extensión a la estructura de la DATA DIVISION del lenguaje COBOL, en orden a proveer estructuras adicionales que reflejan el modelo METASYS. Para ilustrar con algún detalle las características de este lenguaje, se expresa a continuación la base de datos presentada en la Fig. 2.- :

```
DEFINE BASE:  BASE-DE-DATOS-UNIVERSIDAD,BDUN.
     ENTITY:  FACULTAD, KEY=ID-FACULTAD.

              01  REG-FACULTAD.
                  03  ID-FACULTAD                  PIC x(06).
                  03  NOM-FACULTAD                 PIC x(40).
                        '
                        '
                        '

     ENTITY:  ALUMNO, KEY=ID-ALUMNO,
              INDEXED BY =(CEDULA-ALUMNO,NODUP).

              01  REG-ALUMNO.
                  03  ID-ALUMNO                    PIC x(06).
                  03  NOM-ALUMNO                   PIC x(40).
                  03  CEDULA-ALUMNO                PIC 9(10).
                        '
                        '
                        '

     ENTITY:  ASIGNATURA, KEY=ID-ASIGNATURA.

              01  REG-ASIGNATURA
                  03  ID-ASIGNATURA                PIC x(06).
                  03  NOM-ASIGNATURA               PIC x(40).

   RELATION:  ASIGNATURA-OFRECIDA,NULL
              FROM FACULTAD TO ASIGNATURA.
```

```
RELATION:  INSCRIPCION, UNIQUE
           FROM FACULTAD TO ASIGNATURA.

RELATION:  INSCRIPCION,UNIQUE
           BETWEEN ALUMNO AND ASIGNATURA
           INDEXED BY = (FECHA-INSCRIPCION,DUP).
           01  REG-INSCRIPCION.
               03  NUM-INSCRIPCION.                PIC 9(05).
               03  FECHA-INSCRIPCION               PIC 9(06).
               03  RESULT-INSCRIPCION              PIC x.
                        .
                        .
                        .

RELATION:  EVALUACION,HIERARCHICAL
           FROM ASIGNATURA.
           01  REG-EVALUACION.
               03  TIPO-EVALUACION                 PIC x.
                        .
                        .
                        .
```

Puede observarse que METASYS-DDL permite declarar la base (DEFINE BASE), así como cada entidad (ENTITY) y relación (RELATION) por separado. Toda entidad declarada exige la especificación de su clave primaria (KEY), aceptando adicionalmente la especificación de índices secundarios (INDEXED BY) que pueden ser del tipo duplicados (DUP) o del tipo que no aceptará duplicación de contenido (NODUP).

Las relaciones, a su vez, podrán ser declaradas en METASYS-DDL indicando su calidad (UNIQUE, NONUNIQUE, NULL, HIERARCHICAL) junto al tipo y definición de las entidades entre las cuales establece alguna asociación (FROM/TO para 1:n y BETWEEN/AND para n:m). El contenido de cada objeto así declarado, se expresa en términos normales del lenguaje, constituyendo METASYS-DDL una simple extensión de la DATA DIVISION. Esta especificación, sin embargo, no es incorporada directamente a ningún programa de aplicación, sino que pasa a formar parte del DICCIONARIO DE DATOS de la instalación, desde donde será posteriormente extraída durante el preprocesamiento de los programas de aplicación que contengan expresiones del lenguaje METASYS-DML.

Para la manipulación de la información almacenada, METASYS provee el lenguaje METASYS-DML que opera bajo la modalidad de registro-a-la-vez (record-at-a-time), inspirado principalmente en aquellos lenguajes desarrollados para el tratamiento de modelos de redes. Este lenguaje está formado por un conjunto de verbos que configuran una extensión de la PROCEDURE DIVISION del lenguaje COBOL, agregando la claúsula SELECTDB en la

ENVIRONMENT DIVISION para declarar la presencia de una base de datos dada en el programa.

Con el propósito de actualizar la base de datos, METASYS-DML ofrece los verbos:

STORE
MODIFY y
DELETE

que permiten, respectivamente: ingresar, modificar o eliminar una entidad o una relación. Cada una de estas operaciones es llevada a cabo comprobando su completa consistencia con el modelo definido. Es imposible, por ejemplo, eliminar una entidad que tiene relaciones activas dependientes, o modificar una relación alterando el contenido de un índice secundario que derive en una violación de su condición de no duplicado.

Además de los verbos de actualización, METASYS-DML provee un conjunto de expresiones destinadas a permitir la navegación a través de las estructuras del modelo definido. Estas expresiones constituyen variaciones del verbo FIND, básicamente sustentadas en el siguiente repertorio de sentencias:

FIND
FIND FIRST
FIND NEXT
FIND LAST
FIND PRIOR

y están destinadas a establecer accesos a los datos almacenados a través de los múltiples caminos que hayan sido definidos para un modelo en particular.

El programador de aplicaciones construirá su programa en lenguaje COBOL, incorporando los verbos de METASYS-DML al código de la PROCEDURE DIVISION. Una fase de pre-procesamiento del programa, que se sustenta en el DICCIONARIO DE DATOS de la instalación, convertirá toda expresión de METASYS-DML en una invocación (CALL) a alguna función apropiada de los módulos componentes del sistema administrador básico. Dado que el pre-procesador "conoce" detalladamente el modelo de datos definidos, será capaz de disgnosticar toda inconsistencia semántica de las expresiones METASYS-DML incorporadas al programa.

4.- LENGUAJE DE CONSULTA MTSQL

La disponibilidad de un lenguaje de las características de METASYS-DML, ofrece un conjunto de facilidades al programador de a-

plicaciones en el manejo de las estructuras de la base de datos definida, con la ventaja adicional de una permanente protección de la consistencia del modelo. Sin embargo, presenta una seria limitante, por cuanto toda consulta a la base de datos deberá ser únicamente respondida en base a programas especialmente construídos para ello.

Esta limitante puede ser subsanada mediante la presencia de un lenguaje que sea capaz de expresar el objetivo de la consulta, sin especificar la forma de llevar a cabo el procedimiento (non-procedural). Para ello, ha sido desarrollado el lenguaje MTSQL que opera sobre modelos definidos en el sistema METASYS, bajo una modalidad de archivo-a-la-vez (set-at-a-time).

El lenguaje MTSQL se inspiró primitivamente en aquellos lenguajes concebidos para el manejo de modelos relacionales, específicamente SEQUEL [9] , en atención a la propiedad que presenta cualquier base de datos de ser observable como compuesta por un conjunto de tablas relacionales.

En efecto, si cada entidad o relación del modelo METASYS se considera como una tabla relacional, explicitando la existencia en cada relación de los campos claves de las entidades que vincula, es posible aplicar a este nuevo modelo las funciones clásicas del álgebra relacional (SELECT,JOIN,PROJECT).

Sin embargo, en consideración a la complejidad que presenta la expresión en SEQUEL de consultas que requieran sucesivas aplicaciones de la función JOIN, se ha optado por una forma de lenguaje que, manteniendo cierta similitud con la expresión relacional, aproveche las particulares características del modelo METASYS, principalmente aquellas que se derivan del caracter binario de las relaciones. Asimismo la implementación de MTSQL ha excluído la función PROJECT en atención al costo en recursos de hardware y tiempo de proceso que conlleva. De manera que una consulta en MTSQL se limita a:

- expresar la tabla objetivo,
- expresar el camino (PATH), en términos de elementos del modelo, que lleva implícita la función JOIN, y
- expresar los predicados que establecen las restricciones para cada elemento incluído en el camino.

Es así como tomando el ejemplo de la base de datos de la Fig. 2.-, la consulta: "EXHIBIR TODOS LOS ALUMNOS QUE SE ENCUENTRAN INSCRITOS EN LA ASIGNATURA "CCI400", se expresa en MTSQL como sigue:

```
EXHIBIT  :  (ID-ALUMNO, NOM-ALUMNO)

PATH     :  ASIGNATURA (ID-ASIGNATURA = "CCI400')
```

INSCRIPCION
ALUMNO

Esta consulta será satisfecha a través de la pantalla (EXHIBIT) en la siguiente forma:

```
ID-ALUM  NOM-ALUM
---------+--------------------------
         |
037543   | ABARCA MORALES ALBERTO
087583   | ZAMORA BASCUÑAN PEDRO
'        | '
'        | '
'        | '
```

Obviamente, para una consulta como la presentada no se aprecia una diferencia substancial, en cuanto a complejidad de expresión, respecto a SEQUEL:

```
SELECT   ID-ALUMNO, NOM ALUMNO
FROM     ALUMNO
WHERE    ALUMNO.ID-ALUMNO=INSCRIPCION.ID-ALUMNO
AND      ID-ASIGNATURA="CCI400"
```

Sin embargo, en una consulta de mayor complejidad es posible destacar la ventaja de MTSQL. Así por ejemplo, si se quiere obtener "LAS ALUMNAS MAYORES DE 21 AÑOS INSCRITAS EN ASIGNATÚRAS OFRECIDAS POR LA FACULTAD "LEYES", su especificación en MTSQL queda como sigue:

```
EXHIBIT  (ID-ALUMNO, NOM,ALUMNO,EDAD-ALUMNO)

PATH     FACULTAD (ID-FACULTAD=LEYES )
         ASIGNATURA-OFRECIDA
         ASIGNATURA
         INSCRIPCION
         ALUMNO (SEXO-ALUMNO="F" AND
                 EDAD-ALUMNO>"21")
```

en tanto en SEQUEL tendría la siguiente forma:

```
SELECT   ID-ALUMNO, NOM-ALUMNO,EDAD-ALUMNO
FROM     ALUMNO
WHERE    SEXO-AL="F" AND EDAD-ALUMNO > 21
         AND ID-ALUMNO IN
         (SELECT ID-ALUMNO FROM INSCRIPCION   WHERE
         INSCRIPCION.ID-ASIGNATURA=ASIGNATURA-OFRECIDA.ID-ASIGNATURA
         AND
         ID-FACULTAD="LEYES")
```

MTSQL considera además, para aquellas consultas que implican restricciones sobre atributos no indizados o que no consideran restricciones, la posibilidad de definir el "punto de entrada", a través de un índice al camino de acceso, lo cual implica expresar de paso un ordenamiento. Así por ejemplo, la consulta: "LISTAR TODOS LOS ALUMNOS QUE VIVEN FUERA DE VALPARAISO", puede ser expresada:

```
LIST    :   (ID-ALUMNO, CEDULA-ALUMNO, NOMBRE-ALUMNO, CIUDAD-ALUMNO)

PATH    :   ALUMNO (CIUDAD-ALUMNO NOT="VALPARAISO")

KEY     :   CEDULA-ALUMNO, ASCENDING
```

En este caso el recorrido de la entidad se realizará por el índice definido sobre CEDULA-ALUMNO, obteniéndose el listado ordenado por este atributo ascendentemente (ASCENDING), pudiendo definirse también la alternativa contraria (DESCENDING).

5.- CONCLUSIONES

La implementación de MTSQL, de acuerdo a lo descrito en el presente trabajo, constituye una versión simplificada de un lenguaje de consulta, percibiéndose que se han omitido opciones de especificación que hubiesen podido incorporarse sin gran esfuerzo al lenguaje. Este es el caso de la especificación de operaciones aritméticas sobre ciertos atributos asociados a la consulta, o la especificación de ordenamientos diferentes a aquellos en que la tabla objetivo es producida. Para justificar estas simplificaciones, es necesario indicar que a lo largo del proyecto se ha adoptado permanentemente como criterio de desarrollo, el dotar al sistema únicamente de aquellas opciones absolutamente necesarias al usuario medio de instalaciones reducidas, intentando compatibilizar las múltiples posibilidades del modelo con la carencia real de recursos computacionales del medio señalado. Asimismo, ha existido una tendencia predominante a comprobar en el corto plazo la factibilidad y operatividad del modelo, a costa de la exclusión de funciones tal vez muy interesantes.

De acuerdo a esta estrategia de desarrollo, la totalidad de los componentes de METASYS han superado satisfactoriamente la etapa de prueba, encontrándose en normal funcionamiento una serie de aplicaciones que han permitido demostrar no solo la adaptabilidad práctica del sistema a los problemas del usuario final, sino además su simplicidad de manejo para los propósitos de análisis, diseño y programación.

A su vez, MTSQL se encuentra en la fase de experimentación, con el objeto de verificar las hipótesis de trabajo adoptadas res-

pecto de la conducta del usuario frente al lenguaje y frente a sus productos de información. Hasta este instante, no se perciben actitudes del usuario sustancialmente diferentes a aquellas manifestadas respecto de otros lenguajes de consulta como SEQUEL. Sin embargo, las restantes fases del proyecto que se comienzan a abordar: incorporación de nuevas funciones y soporte de lenguaje natural, indudablemente ayudarán a despejar las incógnitas que aún se mantienen en torno a este aspecto de la relación hombre-máquina.

Finalmente, cabe destacar los beneficios subsidiarios que aporta un proyecto de esta naturaleza en la forma de un enriquecimiento de la tecnología local en el área de ingeniería de software, puesto que durante su transcurso ha sido necesario desarrollar y consolidar una serie de herramientas computacionales, así como un conjunto de metodologías, que son sistemáticamente divulgadas por la Universidad en el medio industrial.

REFERENCIAS

[1] A. Morales, L. Barra
System Development Techniques for Small and Medium Size Installations
Proceeding of The 15th SIGCPR Conference
ACM, New York, 1977

[2] A. Morales, O. Mascaró
Economía y Estandarización en la Producción de Software
Proceedings PANEL'78
Universidad Católica de Valparaíso, 1978

[3] L. Barra, A. Morales
Diagramas de Flujo para Programación Estructurada
Proceedings PANEL'79
Universidad Católica de Valparaíso, 1979

[4] A. Morales
Comments on Small Database Implementation
5th International Conference on Very Large Data Bases
Rio de Janeiro, 1979

[5] A. Morales
Implementación de Bases de Datos en Instalaciones Reducidas
Proceedings PANEL'79
Universidad Católica de Valparaíso, 1979.

[6] A. Morales, O. Mascaró, R. Morales
METASYS: Herramientas de Software para la Implementación
de Bases de Datos en Instalaciones Reducidas
Proceedings PANEL'80
Universidad Simón Bolivar, Caracas, 1980.

[7] P. Chen
The Entity-Relationship Model - Toward a
Unified View of Data
ACM Transaction on Database System
New York, 1976

[8] E.F. Codd
Relational Completness of Data Base Sublanguages
Database Systems
Convant Computer Science Symposia Series
Prentice Hall, 1972

[9] M.M. Astrahan, D.D. Chamberlain
Implementation of a Structured English
Query Language
R J 1464 (# 22484) IBM Research Lab.
San José, CA, 1974.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

MECANIZACION DE UNA BIBLIOTECA UTILIZANDO UNA BASE DE DATOS RELACIONAL

J. L. Becerril
R. Casajuana
F. Valer
Centro de Investigación UAM-IBM
Paseo de la Castellana 4, Madrid-1, España

J. Muñoz
Facultad de Informática
Universidad Politécnica de Madrid, España

RESUMEN

En este trabajo se describe un sistema diseñado e implementado en el Centro de Investigación UAM-IBM para la mecanización de una biblioteca. El sistema permite realizar las operaciones típicas de mantenimiento y gestión de una biblioteca, así como seleccionar información de modo conversacional.

El sistema esta implementado en PL/I y utiliza como sistema de gestión de base de datos el Sistema R, un prototipo experimental de base de datos relacional desarrollado en el Laboratorio de Investigación de IBM en San José (USA). Los programas de PL/I que realizan las operaciones citadas incluyen sentencias de SQL, lenguaje no procedural que constituye la interfase externa del Sistema R.

INTRODUCCION

En el área de las bases de datos relacionales (BDR), uno de los aspectos más importantes es el de los lenguajes para su tratamiento (DAT75). En ellos podemos distinguir los siguientes aspectos:

- Interrogación (interacción directa del usuario con el sistema de gestión de la BD).
- Manipulación de datos (inserción, eliminación y actualización de tuplas).

- Definición de datos (creación de relaciones y otras estructuras derivadas).
- Control de datos (manejo de transacciones, integridad de los datos y autorizaciones).

Considerando el contexto de la gestión de una biblioteca, y dado el entorno en que se encuentran sus usuarios, es evidente que el lenguaje de tratamiento de la BDR deberá ser en todas sus facetas un lenguaje cómodo de utilizar y que no requiera profundos conocimientos de informática. El modo elegido en nuestro sistema es el de guiar al usuario por medio de menús, evitando así los problemas de utilización de comandos con una sintaxis fija. Otra de las decisiones básicas que se han tomado es que dichos menús engloben todas las funciones necesarias para el tratamiento de la BDR.

El sistema presenta al usuario en la pantalla una serie de menús encadenados donde se le ofrecen las diversas opciones a las que tiene acceso según sea un usuario privilegiado o no. En la zona inferior de la pantalla es donde el usuario escribe los datos exigidos por el sistema según la operación que desee realizar. En la parte superior el usuario tiene presentes los datos que ya ha suministrado al sistema. Cuando el sistema requiera información adicional la solicitará utilizando la zona central de la pantalla. Los errores detectados por el sistema se indican asimismo en la parte central de la pantalla.

Los mensajes de error que da el sistema son en general autoexplicativos. Con el fin de que el sistema sea cómodo de utilizar, los errores no producen ruptura de la comunicación con el usuario.

Los resultados son enviados a la impresora cuando previsiblemente son de gran volumen, como es el caso de listados completos de la biblioteca. En la búsqueda selectiva el usuario puede decidir donde obtener la salida, es decir, por impresora o por la pantalla, disponiendo en este último caso de facilidades de edición y/o copia de la pantalla.

El sistema prepara un fichero, a modo de diario, con la secuencia de operaciones realizadas, que imprime al concluir la sesión, constituyendo una importante ayuda a la hora de una posterior verificación.

Para la realización del sistema se ha utilizado como lenguaje el PL/I y como sistema de gestión de BD el Sistema R (AST76), un prototipo experimental desarrollado en IBM Research (San Jose,USA). Los programas de PL/I que realizan las operaciones citadas incluyen sentencias de SQL (CHA76), lenguaje que constituye la interfase externa del Sistema R. La comunicación con el usuario se realiza a traves de una pantalla IBM 3270.

DISEÑO DE LA BASE DE DATOS

La información considerada en este sistema bibliográfico se han organizado en relaciones según el esquema conceptual que muestra la tabla siguiente:

RELACION	DOMINIO	TIPO
AUTORES	NUM	BIN FIXED(15)
AUTORES	AUTOR	CHAR(40) VAR
AUTORES	TIPO	CHAR(1)
LIBROS	NUM	BIN FIXED(15)
LIBROS	F_EDICION	BIN FIXED(15)
LIBROS	N_PAG	BIN FIXED(15)
LIBROS	N_EJEMPLARES	BIN FIXED(15)
LIBROS	EDITORIAL	CHAR(40) VAR
LIBROS	TITULO	CHAR(130) VAR
LIBROS	I_SIG	BIN FIXED(15)
PRESTAMOS	NUM	BIN FIXED(15)
PRESTAMOS	EJEMPLAR	BIN FIXED(15)
PRESTAMOS	I_USUARIO	BIN FIXED(15)
PRESTAMOS	FECHA	BIN FIXED(31)
SIGNATURAS	I_SIG	BIN FIXED(15)
SIGNATURAS	SIG1	CHAR(5) VAR
SIGNATURAS	SIG2	CHAR(5) VAR
SIGNATURAS	MATERIA	CHAR(50) VAR
USUARIOS	I_USUARIO	BIN FIXED(15)
USUARIOS	NOMBRE	CHAR(30) VAR
USUARIOS	APELLIDO1	CHAR(30) VAR
USUARIOS	APELLIDO2	CHAR(30) VAR
USUARIOS	LOCALIZACION	CHAR(3)
USUARIOS	COMENTARIOS	CHAR(100) VAR

Tabla 1 . Esquema conceptual de la BD.

Se ha tratado de conseguir un equilibrio entre:

- Minimizar el número de relaciones, ya que lo contrario implicaría la necesidad de gran número de "joins" para realizar cualquier operación, lo cual en general es costoso en tiempo.
- Evitar duplicaciones innecesarias de dominios que, además de aumentar el tamaño de la BD, hacen difícil y costoso el mantener la integridad.
- Prever la posibilidad de añadir nueva información sobre los libros, sin que esto suponga la reestructuración de la BD, lo cual equivale a escoger un diseño facilmente extensible.

Observemos que salvo en las relaciones SIGNATURAS y USUARIOS, en todas las restantes hay un dominio común (NUM) correspondiente al número del libro, dominio que permite realizar "joins" entre esas relaciones. Para la relación SIGNATURAS el dominio que permite hacer "joins" con las restantes es I_SIG, el identificador numérico de la signatura, mientras que para la USUARIOS es I_USUARIO, el identificador numérico de usuario. La figura 1 muestra estas conexiones; las flechas indican posibles "joins".

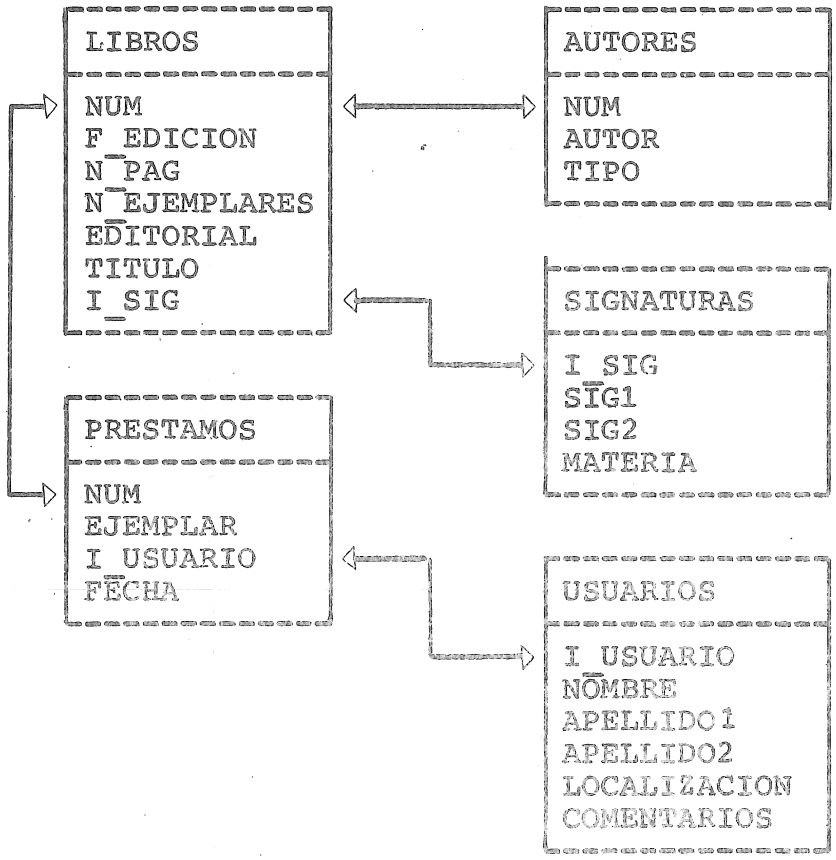


Figura 1 . Posibles conexiones entre las relaciones

Los únicos dominios que no tienen nombres autoexplicativos son TIPO en la relación AUTORES y el par SIG1, SIG2 en SIGNATURAS. TIPO contempla el hecho de que un libro pueda tener uno o varios autores; en el primer caso este dominio contiene una 'U', mientras que en el segundo el primer autor tiene en este dominio una 'P', y los demas el valor ' ' . El par de dominios SIG1, SIG2 contienen la signatura del libro partida en dos campos, entendiendo por signatura un identificador único para cada libro asignado según la materia principal que contenga.

Es de destacar que, dada la flexibilidad de las BDR, la extensión de este esquema, incluyendo nuevos dominios (en el

caso de que se desee almacenar nuevos datos sobre los libros) no provocaría grandes complicaciones en la estructura de los programas y por supuesto, no requeriría una nueva estructuración y carga de la BD. Caso de desear añadir información que sea única por libro (ej. precio) sólo será preciso expandir la relación libros con los dominios necesarios. En el caso de que a cada libro corresponda más de un elemento de información (ej. palabras clave), deberá crearse una nueva relación conectada con LIBROS mediante el dominio NUM que contenga la información deseada, análogamente a lo que ocurre con las relaciones AUTORES o SIGNATURAS.

Con el fin de que las interrogaciones sean mas eficientes, se han creado índices (ordenaciones por valor) sobre los dominios mas significativos en las operaciones mas típicas. Existe la posibilidad adicional de añadir o cambiar los índices, con lo que se consigue "adaptar" la BD a las aplicaciones más usuales o conceder un trato privilegiado a ciertas operaciones.

Se han creado índices en el dominio NUM de las relaciones AUTORES, LIBROS y PRESTAMOS, y en I SIG de las relaciones LIBROS y SIGNATURAS, en I USUARIO de las relaciones PRESTAMOS y USUARIOS, con la idea de disminuir el costo de los "joins" entre las relaciones de mayor cardinalidad. Con el fin de que algunas busquedas típicas sean más rápidas, también se han creado índices en el dominio AUTOR de la relación AUTORES y en el par SIG1, SIG2 de la relación SIGNATURAS.

OPCIONES DEL SISTEMA

El sistema permite realizar operaciones que podemos agrupar en:

- Mantenimiento y gestión de la biblioteca: entrada de libros, gestión de préstamos, archivo de usuarios, obtención de listados de la biblioteca completa ordenados por autores, signaturas, usuarios, etc.

- Búsqueda conversacional, seleccionando entre la información almacenada según ciertos criterios elementales o combinados (fecha de edición, autor, etc.).

Los dos modos de operación implican dos tipos de usuarios diferentes; el primero es el administrador de la biblioteca, mientras que el segundo es el auténtico usuario final de un sistema bibliográfico. No obstante, se ha tratado de que la forma de utilización del sistema sea igualmente sencilla en cualquiera de los casos sin requerir apenas aprendizaje, siempre preservando los niveles de seguridad imprescindibles en cuanto a acceso a la actualización y mantenimiento de la integridad de los datos almacenados.

El sistema se invoca tecleando la palabra LIBROS. Aparece el primer menú que ofrece al usuario las posibilidades de:

- Actualización de la biblioteca (ACT).
- Listados de la biblioteca (LIS).
- Busqueda coversacional (BUS).

Existen otras opciones que no aparecen en el menú para uso del administrador de la base de datos, que permiten operaciones privilegiadas como creación de índices, obtención de información sobre el catálogo de la BD, etc.

Las operaciones típicas de mantenimiento y gestión se realizan escogiendo en el primer menú alguna de las dos opciones ACT ó LIS. Desde la opción ACT, las operaciones posibles son:

- Entrada de libros (LIB).
- Actualización del catálogo de signaturas (CAT).
- Actualización de bajas (BAJ).
- Actualización de préstamos (PRE).
- Devolución de libros (DEV).
- Actualización de la lista de usuarios (AUS).

En función de la opción escogida el sistema va solicitando al usuario los datos necesarios para su posterior proceso y almacenamiento en la BD.

Desde la opción LIS, las operaciones posibles son:

- Listado de la biblioteca completa por autores (AUT).
- Listado de la biblioteca completa por signaturas (LSG).
- Catálogo de signaturas (CSG).
- Listado de la biblioteca por números (NUM).
- Listado de préstamos por usuarios (LPU).
- Listado de préstamos por números (LPN).
- Listado de usuarios (USU).
- Todos los anteriores (TODOS).

Las operaciones relativas a búsqueda conversacional se realizan escogiendo en el primer menú la opción BUS. Desde esta opción las operaciones posibles son:

- Obtención de libros que contengan en su título una cierta palabra clave (TITC).
- Obtención de libros con un autor o autores determinados (AUTC).
- Obtención de libros cuyo año de edición sea N, mayor que N, o entre N y M (FEDC).
- Obtención de libros con una cierta signatura (SIGC).
- Obtención de libros que contengan en su materia una cierta palabra clave (MATC).
- Obtención de libros que pertenezcan a una editorial (EDIC).

- Mezcla de las posibilidades anteriores (MEZC).

IMPLEMENTACION DE LAS OPCIONES

En esta sección vamos a describir cómo se ha realizado la implementación de las distintas opciones del sistema. Todas ellas tienen como parte básica sentencias de SQL (interfase externa del Sistema R) incluidas en programas PL/I. En la descripción de las sentencias de SQL utilizaremos nombres que comienzan por el signo "$" cuando hagamos referencia a variables del programa, para no confundirlas con los mismos nombres utilizados como identificadores de dominios.

LIB. Entrada de libros.

Se trata de almacenar en la base de datos la información correspondiente a un libro nuevo. En primer lugar, conocida la signatura (el par $SIG1, $SIG2), se localiza el identificador de signatura ($ISIG) que le corresponde.

```
SELECT I_SIG INTO $ISIG
FROM   SIGNATURAS
WHERE  SIG1=$SIG1 AND SIG2=$SIG2
```

Si no existe la signatura se calcula el nuevo identificador y se insertan en la relación SIGNATURAS los datos correspondientes.

```
SELECT MAX(I_SIG) INTO $ISIG
FROM   SIGNATURAS

INSERT INTO SIGNATURAS :
  <$ISIG+1,$SIG1,$SIG2,$MAT>
```

A continuación se inserta el resto de los datos en las tres relaciones involucradas (AUTORES, PRESTAMOS y LIBROS). Siempre se supone que un libro nuevo no está prestado, es decir $IUSU=1.

```
INSERT INTO AUTORES :
  <$NUM,$AUT,$TIPO>

INSERT INTO PRESTAMOS :
  <$NUM,$EJ,$IUSU,$FECHA>

INSERT INTO LIBROS :
  <$NUM,$ISIG,$FEDI,$NPAG,$NEJ,$EDI,$TIT>
```

En el caso de querer añadir un ejemplar de un libro ya existente registrado con el número $NUM, sólo es necesario conocer el número del último ejemplar, actualizar el número de

ejemplares en la relación LIBROS e insertar una nueva tupla en PRESTAMOS indicando que existe un nuevo ejemplar del libro citado.

```
SELECT MAX(EJEMPLAR) INTO $EJE
FROM   PRESTAMOS
WHERE  NUM=$NUM

UPDATE LIBROS
SET    N_EJEMPLARES = N_EJEMPLARES + 1
WHERE  NUM=$NUM

INSERT INTO PRESTAMOS :
  < $NUM,$EJE,$IUSU,$FECHA >
```

CAT. Actualización del catálogo de signaturas.

Para añadir una signatura nueva, en primer lugar se calcula el identificador que le va a corresponder ($ISI + 1) y después se inserta una tupla en la relación SIGNATURAS.

```
SELECT MAX(I_SIG)
INTO   $ISI
FROM   SIGNATURAS

INSERT INTO SIGNATURAS :
< $ISI+1,$SIG1,$SIG2,$MAT >
```

BAJ. Actualización de bajas.

En primer lugar se obtienen los datos correspondientes al libro con número $NUM y ejemplar $EJE.

```
SELECT PRESTAMOS.I_USUARIO,NOMBRE,APELLIDO1,APELLIDO2
INTO   $IUSU,$NOM,$AP1,$AP2
FROM   PRESTAMOS,USUARIOS
WHERE  PRESTAMOS.I_USUARIO = USUARIOS.I_USUARIO
AND    PRESTAMOS.NUM = $NUM
AND    PRESTAMOS.EJEMPLAR = $EJE
```

A continuación se elimina la información correspondiente al libro que causa baja.

```
DELETE PRESTAMOS
WHERE  NUM=$NUM
AND    EJEMPLAR=$EJE

DELETE LIBROS WHERE NUM=$NUM

DELETE AUTORES WHERE NUM=$NUM
```

Seguidamente se actualiza el número de ejemplares.

```
UPDATE LIBROS
SET    N_EJEMPLARES = N_EJEMPLARES-1
WHERE  NUM=$NUM
```

PRE, DEV. Actualización de préstamos o devoluciones.

En ambos casos, tras detectar si el libro a prestar (devolver) está en la biblioteca (prestado a ese usuario), se debe actualizar la relación PRESTAMOS de modo que refleje la nueva situación del libro, es decir, se cambia la fecha y la identificación del usuario.

```
UPDATE PRESTAMOS
SET    I_USUARIO = $I_USU,
       FECHA     = $FECHA
WHERE  NUM       = $NUM_PRE
AND    EJEMPLAR  = $EJE_PRE
```

AUS. Actualización de usuarios.

En este caso, tras calcular la identificación que le va a corresponder al nuevo usuario, se inserta una tupla en la relación USUARIOS con los datos pertinentes.

```
SELECT MAX(I_USUARIO)
INTO   $IUS
FROM   USUARIOS

INSERT INTO USUARIOS:
  <$IUS+1,$NOM,$APL1,$APL2,$LOC,$COM>
```

AUT. Listado por autores

La sentencia siguiente:

```
LET C1 BE
  SELECT AUTOR,TITULO,LIBROS.NUM,N_EJEMPLARES
         EDITORIAL,F_EDICION,N_PAG,SIG1,SIG2
  INTO   $AUT,$TIT,$NUM,$NEJ,$EDI,$FEDI,$NPAG,$SIG1,$SIG2
  FROM   LIBROS,AUTORES,SIGNATURAS
  WHERE  LIBROS.NUM = AUTORES.NUM
  AND    LIBROS.I_SIG = SIGNATURAS.I_SIG
  ORDER BY 1,2
```

define un cursor, que en sucesivos "fetch" pondrá en las variables asociadas los datos necesarios en orden alfabético de autores y títulos.

LSG. Listado por signaturas.

El cursor S1 nos suministrará el identificador I_SIG de cada signatura, además de la materia asociada a cada una de ellas.

```
LET S1 BE
  SELECT I_SIG,SIG1,SIG2,MATERIA
  INTO   $ISIG,$SIG1,$SIG2,$MAT
  FROM   SIGNATURAS
  ORDER BY 2,3
```

Para cada valor de la variable $ISIG se define el cursor C1, mediante el cual se obtienen todos los libros con esa signatura (condición I_SIG=$ISIG). La condición sobre el dominio TIPO es necesaria para limitar el resultado al primer autor de cada libro.

```
LET C1 BE
  SELECT AUTOR,TITULO,LIBROS.NUM,N_EJEMPLARES,
         EDITORIAL,F_EDICION,N_PAG,TIPO
  INTO   $AUT,$TIT,$NUM,$NEJ,$EDI,$FEDI,$NPAG,$TIPO
  FROM   LIBROS,AUTORES
  WHERE  LIBROS.NUM = AUTORES.NUM
  AND    LIBROS.I_SIG = $ISIG
  AND    AUTORES.TIPO  ≠ ' '
  ORDER BY 1,2
```

CSG. Catálogo de signaturas.

El siguiente cursor proporciona los identificadores y la significación para cada signatura:

```
LET C1 BE
  SELECT SIG1,SIG2,MATERIA
  INTO   $SIG1,$SIG2,$MAT
  FROM   SIGNATURAS
  ORDER BY 1,2
```

NUM. Listado por números.

Este caso es muy similar a la segunda parte del LSG:

```
LET C1 BE
  SELECT LIBROS.NUM,TITULO,N_EJEMPLARES,
         SIG1,SIG2,AUTOR,TIPO
  INTO   $NUM,$TIT,$NEJ,$SIG1,$SIG2,$AUT,$TIPO
  FROM   LIBROS,AUTORES,SIGNATURAS
  WHERE  LIBROS.NUM = AUTORES.NUM
  AND    LIBROS.I_SIG = SIGNATURAS.I_SIG
  AND    AUTORES.TIPO  ≠ ' '
  ORDER BY 1
```

LPU. Listado de préstamos por usuarios.

El cursor C1 nos suministra los datos de cada usuario, además de su identificador (I_USUARIO), con excepción del usuario número 1 que corresponde a la propia biblioteca.

```
LET C1 BE
  SELECT I_USUARIO,NOMBRE,APELLIDO1,
         APELLIDO2,LOCALIZACION,COMENTARIOS
  INTO   $IUSU,$NOM,$AP1,$AP2,$LOC,$COM
  FROM   USUARIOS
  WHERE  I_USUARIO  ≠ 1
  ORDER BY 3
```

La sentencia siguiente calcula el número de tuplas de la relación PRESTAMOS que verifican la condición I_USUARIO=$IUSU, es decir, para cada identificador de usuario calcula cuantos libros tiene préstados.

```
  SELECT COUNT(*) INTO $N
  FROM   PRESTAMOS
  WHERE  I_USUARIO = $IUSU
```

LPN. Listado de préstamos por números.

Este cursor selecciona todos los libros que aparecen en la relación PRESTAMOS con un identificador de usuario distinto del de la biblioteca (≠1).

```
LET C1 BE
  SELECT LIBROS.NUM,TITULO,SIG1,SIG2,
         NOMBRE,APELLIDO1,APELLIDO2,EJEMPLAR,FECHA
  INTO   $NUM,$TIT,$SIG1,$SIG2,$NOM,$AP1,$AP2,$EJE,$FECHA
  FROM   LIBROS,SIGNATURAS,USUARIOS,PRESTAMOS
  WHERE  LIBROS.NUM = PRESTAMOS.NUM
  AND    LIBROS.I_SIG = SIGNATURAS.I_SIG
  AND    PRESTAMOS.I_USUARIO = USUARIOS.I_USUARIO
  AND    PRESTAMOS.I_USUARIO  ≠ 1
  ORDER BY 1
```

USU. Listado por usuarios.

En este caso el cursor U1 obtiene los datos de cada uno de los usuarios.

```
LET U1 BE
  SELECT I_USUARIO,NOMBRE,APELLIDO1,APELLIDO2
  INTO   $IUSU,$NOM,$AP1,$AP2
  FROM   USUARIOS
  WHERE  I_USUARIO  ≠ 1
  ORDER BY 3
```

El cursor C1, para cada valor de $IUSU, obtiene todos los datos de los libros prestados a ese usuario.

```
LET C1 BE
  SELECT LIBROS.NUM,TITULO,AUTOR,TIPO,FECHA,EJEMPLAR
  INTO   $NUM,$TIT,$AUT,$TIPO,$FECHA,$EJE
  FROM   LIBROS,AUTORES,PRESTAMOS
  WHERE  LIBROS.NUM = PRESTAMOS.NUM
  AND    LIBROS.NUM = AUTORES.NUM
  AND    PRESTAMOS.I_USUARIO = $IUSU
  AND    AUTORES.TIPO  ≠ ' '
  ORDER BY 5
```

Búsqueda conversacional.

En líneas generales, el proceso a seguir en todos los casos de la búsqueda conversacional consiste en encontrar los números de los libros que verifican las condiciones indicadas por el usuario. Posteriormente, es fácil obtener los datos restantes de los libros involucrados mediante sentencias similares a las anteriormente descritas.

Para encontrar los números de los libros seleccionados hay tres casos muy similares, en los que la sentencia será del tipo:

```
LET C1 BE
  SELECT  NUM     INTO  $NUM
  FROM    LIBROS
  WHERE   .........
```

donde la condición será para la opción:

TITC: TITULO LIKE '%xx...xx%'

FEDC: FECHA = $N (ó <,>)
      FECHA BETWEEN $N AND $M

EDIC: EDITORIAL = $EDI

Para AUTC los números seleccionados se obtienen mediante el cursor:

```
LET C1 BE
  SELECT NUM INTO $NUM
  FROM   AUTORES
  WHERE  AUTOR IN ($A1,$A2, ...,$An)
```

o condiciones similares.

En el caso de SIGC, conocida la signatura ($SIG1,$SIG2) se obtiene el identificador de signatura mediante:

```
LET S1 BE
  SELECT I_SIG INTO $ISIG
  FROM   SIGNATURAS
  WHERE  SIG1=$SIG1  AND SIG2=$SIG2
```

Para el caso de MATC el cursor sería:

```
LET S1 BE
  SELECT I_SIG INTO $ISIG
  FROM   SIGNATURAS
  WHERE  MATERIA LIKE $MAT
```

teniendo $MAT el valor adecuado.

En cualquiera de estos dos últimos casos, una vez conocido el valor o valores de $ISIG, se define el siguiente cursor:

```
LET C1 BE
  SELECT NUM INTO $NUM
  FROM LIBROS
  WHERE I_SIG IN ($ISIG1, ...,$ISIGn)
```

El caso MEZC es una combinación de todos los anteriores.

Citemos como detalle de interés que, salvo el programa asociado a la opción MEZC, todos los demás están preprocesados y preparados para su ejecución, es decir, con su correspondiente módulo de acceso a la base de datos generado, con la consiguiente ventaja en lo referente a tiempo de ejecución.

REFERENCIAS

(AST76) Astrahan M.M. y otros (1976), System R: A Relational Approach to Database Management, ACM Trans. on Database Systems, Vol. 1, No. 2, Junio 1976.

(CHA76) Chamberlin D.D. y otros (1976), SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control, Research Report RJ1798, IBM Research Lab., San Jose, Cal.,USA. Junio 1976.

(DAT75) Date C.J. (1975), An Introduction to Data Base Systems, Addison-Wesley 1975.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

AUTOMATIZACION DEL PROCESO DE NORMALIZACION DE UNA BASE DE DATOS RELACIONAL

J. L. Becerril
R. Casajuana
F. Valer
Centro de Investigación UAM-IBM
Paseo de la Castellana 4, Madrid-1, España

J. Guizán
Facultad de Informática
Universidad Politécnica de Madrid, España

RESUMEN

Este trabajo describe un sistema para simplificar (y en algunos casos automatizar) el proceso de normalización de una base de datos relacional. Este sistema, ademas de obtener las posibles normalizaciones de la base de datos en 2NF, 3NF y 4NF, calcula el volumen de la base de datos para las distintas normalizaciones.

En la primera parte de la comunicación se introduce la teoría de normalización de Codd ampliada con la cuarta forma normal de Fagin. En el resto se presenta el tratamiento que reciben las dependencias funcionales y multivaluadas para obtener la base de datos en 2NF, 3NF y 4NF. En el apéndice final se incluye un ejemplo realizado con el sistema propuesto.

INTRODUCCION

Desde 1970, en que E.F. Codd, con su trabajo sobre relaciones n-arias en un contexto de base de datos, plantea los fundamentos de la teoría de bases de datos relacionales (BDR), se han realizado enormes esfuerzos dentro de este campo, tanto a nivel formal como al de diseño e implementación de prototipos.

El concepto de normalización (1) fué introducido por Codd en 1970 rigorizándolo posteriormente en 1971. El motivo de la aparición de este concepto fué la observación de que para idénticos datos, diferentes organizaciones entre los mismos en forma de relaciones presentaban muy distintas propiedades en cuanto a actualización, inserción o eliminación de información. Se trataba pues de adoptar organizaciones de los datos que permitieran que las operaciones sobre las relaciones se realizaran con la máxima simplicidad y manteniendo la integridad de la BDR.

El trabajo que se presenta describe un sistema para simplificar (y en algunos casos automatizar) el proceso de transformación de una base de datos en cada una de las cuatro formas normales.

La lógica general es la siguiente:

Una vez que el diseñador de la base de datos ha introducido, utilizando una sintaxis predefinida, las dependencias funcionales o multivaluadas y los datos de tipo estadístico (tamaño medio de las tuplas, número medio de apariciones de distinto valor por dominio, etc.) el sistema evalúa la base de dependencias funcionales que definen la BDR siguiendo el algoritmo de Beeri(5). En función de esta base de dependencias funcionales se normaliza la BD en 2NF. La obtención de la BD en 3NF se realiza siguiendo un algoritmo debido a Bernstein(6). Posteriormente, a partir de las 3NF, y teniendo en cuenta las dependencias multivaluadas, se normaliza la base en 4NF.

NORMALIZACION

El concepto de normalización fué introducido por Codd en (2) rigorizándolo posteriormente en (4). El motivo de la aparición de este concepto fué la observación de que para idénticos datos, diferentes agrupaciones en relaciones presentaban muy distintas propiedades en cuanto a actualización, inserción, etc.

La teoría que vamos a describir se basa en una serie de formas normales (llamadas primera, segunda, tercera, de Boyce-Codd y cuarta formas normales) que proporcionan sucesivas mejoras en cuanto a las propiedades de modificación de una base de datos (3), (4), (5) y (7).

PRIMERA FORMA NORMAL. Se dice que la relación R es una primera forma normal sii ninguna de sus tuplas tiene elementos que a su vez sean conjuntos. Una relación es no normalizada si no es una primera forma normal.

Dependencia funcional (DF). Sean dos colecciones de atributos A y B de R; se dice que B es funcionalmente dependiente de A, y se escribe RA -> RB, sii en cada instante a cada valor de A en R le corresponde a lo sumo uno de B. En todos los casos en que no haya posible confusión, y con el fin de simplificar la notación, en lugar de RA -> RB escribiremos A -> B.

Superclave. Se dice que K es una superclave si es un conjunto de atributos de una relación del que dependen funcionalmente los restantes.

Clave. Se dice que una superclave es una clave si es una superclave mínima, es decir, si no existe otra superclave que esté contenida en ella.

Atributos primarios. Se dice que un atributo de una relación es primario sii aparece en la clave elegida.

Dependencia funcional total. Sean D y E dos subcolecciones distintas de atributos de R, siendo E funcionalmente dependiente de D. Se dice que E presenta, respecto a D, una dependencia funcional total en R sii E no es funcionalmente dependiente de ningún subconjunto de D distinto del total.

SEGUNDA FORMA NORMAL. Se dice que R es una segunda forma normal sii:

1) Es una primera forma normal.
2) Todo atributo que no sea primario es totalmente dependiente de cada clave de R.

Dependencia transitiva. Consideremos tres colecciones distintas de atributos A, B, C en una relación R cuyo grado es mayor o igual a tres. Supongamos que se verifican las tres condiciones independientes del tiempo siguientes: RA -> RB RB $\not\rightarrow$ RA RB -> RC. Si además se verifican RA -> RC y RC $\not\rightarrow$ RA, se dice que C es transitivamente dependiente de A bajo R.

TERCERA FORMA NORMAL. Diremos que una relación R es una tercera forma normal sii:
1) Es una segunda forma normal.
2) Cada atributo que no sea primario, no es transitivamente dependiente de cada clave de R.

FORMA NORMAL DE BOYCE-CODD. R es BCNF sii es 3NF y para todo par de conjuntos de atributos X,Y ($X,Y \neq \phi$, $X \cap Y = \phi$), si X -> Y entonces X es una superclave de R. Con otras palabras, si algun atributo depende de X, todos los demás también.

Proyección de un atributo. Consideremos tres colecciones disjuntas de atributos X,Y,Z y la relacion R(X,Y,Z). Se define la proyección de Y sobre el punto x,z a:
$Y(x,z) = \{ y \in Y;\ (x,y,z) \in R \}$.

Dependencias Multivaluadas. Se dice que un conjunto de atributos (Y) presenta una dependencia multivaluada (DMV) de otro conjunto de atributos (X) en la relación R(X,Y,Z), y se denota $X \rightarrow\!\!\rightarrow Y$, sii $Y(x,z) = Y(x,z')$ para cada x,z,z' tales que $Y(x,z) \neq \phi$, $Y(x,z') \neq \phi$.

Dependencias Multivaluadas Triviales. Sea R(X,Y); las dependencias multivaluadas triviales son del tipo $X \rightarrow\!\!\rightarrow \phi$, o $X \rightarrow\!\!\rightarrow Y$.

Propiedades de las Dependencias Multivaluadas

- Toda dependencia funcional es multivaluada.
- Si Y presenta una dependencia multivaluada respecto a X en R(X,Y,Z) entonces R es el join de sus proyecciones R'(X,Y) y R"(X,Z).

CUARTA FORMA NORMAL. Se dice que R es una 4NF sii para toda dependencia multivaluada no trivial $X \rightarrow\!\!\rightarrow Y$, entonces $X \rightarrow A$ para todos los atributos A de R.

AUTOMATIZACION DEL PROCESO DE NORMALIZACION

A continuación citaremos algunos resultados referentes a la teoria de la normalización que son de interés en el proceso de su automatización:

- El diseño de una BD depende del conocimiento que se tenga sobre las dependencias funcionales y multivaluadas existentes entre los atributos de los datos. Salvo en el caso de una BD estática, las dependencias no pueden conocerse automáticamente, sino que el diseñador de la BD debe definirlas a partir de la semántica de la información.

- El diseño de la BD en cualquier nivel de normalización no es único y por lo tanto se plantea la posibilidad de escoger una forma normal óptima. Se han realizado numerosos estudios sobre cómo llegar a esta optimización, que no citaremos aquí por quedar fuera del alcance de este trabajo.

- Existe siempre un esquema relacional en 3NF y en 4NF.

- No siempre existe un esquema en BCNF (5).

Dentro de un proceso para automatizar el proceso de normalización anteriormente descrito se pueden distinguir dos

tipos de problemas bien diferenciados. El primero de ellos se refiere al problema teórico de partir de un conjunto de dependencias funcionales y/o multivaluadas y obtener una base de datos en 2NF, 3NF, BCNF o 4NF. El segundo contempla el aspecto práctico relativo a la elección de una posible normalización en función de datos de tipo estadístico (tamaño medio de las tuplas, número medio de apariciones de distinto valor por columnas, cardinalidad y grado de las relaciones, etc.) y características de la BD. En esta comunicación nos vamos a restringir al primer problema, pudiendo verse en (8) una más completa exposición del tema.

Una vez el diseñador de la BD ha suministrado el conjunto de dependencias funcionales y multivaluadas, el sistema realiza las siguientes operaciones:

- Obtención de una base de dependencias funcionales.
- Obtención de la BD en 2NF.
- Obtención de la BD en 3NF.
- Obtención de la BD en 4NF.

ALGORITMO DE EXTRACCION DE DEPENDENCIAS FUNCIONALES.

Amstrong (9) demuestra que el conjunto mínimo de propiedades de las DF, por cuya aplicación se obtienen las DF derivadas son:

- Reflexividad: Si $Y \subseteq X$ entonces $X \rightarrow Y$. Esta propiedad es obvia pues en cada instante cada valor de Y tiene asociado a lo sumo uno en X (el mismo valor).
- Aumentación: Si $X \rightarrow Y$ y $W \subseteq Z$, entonces $X \cup Z \rightarrow Y \cup W$. Propiedad evidente por la anterior y por definición de dependencia funcional.
- Seudotransitividad: Si $X \rightarrow Y$ y $Y \cup Z \rightarrow W$ entonces $X \cup Z \rightarrow W$. Esta regla de sustitución es evidente a partir de la definición de DF.

* Al conjunto de DF que se obtiene por sucesiva aplicación de las reglas anteriores a un conjunto F de DF se le denomina cierre de F^+ .

* Dado un conjunto F de DF se dice que un conjunto H es una cobertura de F si tiene el mismo cierre que F.

* Una cobertura se dice que es no-redundante si no contiene ningún subconjunto que a su vez sea cobertura.

* Una DF $f \in F$ se dice redundante en F si pertenece al cierre de $F - \{f\}$.

* Una cobertura se dice redundante si y solo si contiene DF redundantes.

El problema que nos planteamos en esta sección es: dado un conjunto de DF obtener una cobertura no redundante. Aunque las tres reglas son el conjunto mínimo completo de propiedades de las DF, es conveniente, desde un punto de vista práctico, introducir dos reglas adicionales que son consecuencia inmediata de las anteriores.

- Union: Si X -> Y y X -> Z entonces X -> YZ. En efecto, aplicando la aumentación a X -> Z para Y se tiene XY -> YZ, aplicando a esta regla y a la X -> Y la seudotransitividad se tiene X -> YZ.
- Descomposición: Si X -> Y y $Z \subseteq Y$ entonces X -> Z. Por la reflexividad Y -> Z, como X -> Y aplicando la seudotransitividad queda X -> Z.

De estas últimas dos reglas se obtiene inmediatamente que la dependencia funcional X -> A B ...K, es equivalente al conjunto de dependencias funcionales X -> A, X -> B,... X -> K. Así pues, para probar que una dependencia funcional X -> A B ...K es derivable de un conjunto de DF es suficiente con probar que lo son las X -> A, X -> B,..., X -> K.

Arboles de derivacion. Sea F un conjunto dado de DF donde cada dependencia tiene solo un atributo en su parte derecha (elección factible en función del parrafo anterior). Una derivación de una dependencia funcional f desde F es una secuencia $f_1,...,f_n$ tal que f_n =f y para cada i entre 1 y n se verifica alguna de las siguientes propiedades:

- La DF f_i pertenece a F.
- La DF f_i es el resultado de aplicar alguna de las propiedades de las dependencias funcionales en el conjunto $F - \{f_i\}$.

Con el fin de tratar a nivel práctico el problema de extracción de una cobertura no redundante, vamos a introducir un modelo denominado arbol de derivación, en el que es posible realizar todas las operaciones definidas sobre árboles a la vez que cumplen las propiedades referentes a las DF.

Se define el conjunto de árboles de derivación basados en F por medio de las reglas:

- Si A es un atributo entonces un nodo etiquetado con A es un árbol de derivación basado en F.
- Si T es un árbol de derivación con un nodo hoja etiquetado con A y la dependencia funcional B ... M -> A pertenece a F, entonces el árbol construido a partir de T añadiéndole B,...,M como hijos de A es también un árbol de derivación.
- Un árbol etiquetado es un árbol de derivación solo si es posible obtenerlo por aplicación de un número finito de veces de las dos reglas anteriores.

En este punto ya estamos en condiciones de desarrollar un algoritmo para que, dado un conjunto de dependencias funcionales F y una nueva DF f, podamos determinar si f pertenece a la cierre de F (5).

Consideremos un conjunto de dependencias funcionales F, definidas sobre un conjunto de atributos A,B,...,M. Supongamos que todas las dependencias funcionales que tienen la misma parte izquierda están agrupadas en una sola DF (este hecho no es restrictivo, pues se puede conseguir sin más que aplicar la regla de unión). Representemos el conjunto F por una cadena de pares donde cada par representa una DF y denominemos PD a su parte derecha y PI a la izquierda. Cada parte es un conjunto de atributos que representaremos por números enteros del conjunto 1,2,...,m . La longitud de la representación de F la denotaremos por # F. Sea f : X --> S donde X y S son subconjuntos de {A,B,...,M}. Como cada atributo de f aparece en al menos una DF de F, podemos asumir que # f $\leq$ # F.

El método para probar si f $\in$ F^+ es calcular el conjunto de atributos que son funcionalmente dependientes de X. Sea DEP un conjunto que contiene a esos atributos. Asignemos inicialmente el valor X a DEP, puesto que por reflexividad X es funcionalmente dependiente de X. Para buscar nuevos atributos que se puedan añadir a DEP, seleccionamos una DF, f', que esté en F y cuya parte izquierda esté contenida en DEP pero no su parte derecha. Por seudotransitividad la parte derecha de f' es funcionalmente dependiente de X y puede añadirse a DEP. Podemos continuar seleccionando DF de esta manera, añadiendo su parte derecha a DEP, hasta que ya no se pueda añadir ningún nuevo atributo.

Conceptualmente, DEP contiene un conjunto de atributos cada uno de los cuales es la raiz de un árbol de derivación basado en F y cuyas hojas son las X. Cada vez que encontramos una DF, f', cuya parte izquierda está contenida en DEP, la segunda regla de construcción de los árboles de derivación establece que cada atributo de la parte derecha de f' puede ser la raiz de un arbol de derivación cuyas hojas son X. Sabemos que f$\in$ F^+ si y solo si S esta en DEP.

Una aplicación importante de este algoritmo es la eliminación de DF redundantes en F. Para determinar si una f es redundante en F, aplicamos el algoritmo anterior para probar si f$\in$ $(F - \{f\})^+$. Así pues, un subconjunto de F que sea cobertura no redundante puede ser calculado con el siguiente procedimiento: Hacer G = F, un bucle para cada f F en el que si f$\in$ $(G - \{f\})^+$ se haga G = G -{f}. El valor final de G dependerá en general del orden en que se han seleccionado las DF de F. Además G será por construcción un subconjunto de F, aunque este no sea un requerimiento de cobertura no redundante.

ALGORITMO DE OBTENCION DE LA BASE DE DATOS EN 2NF.

Una vez se ha obtenido la base de DF y eliminado las redundantes, se evaluan las claves de las futuras relaciones en 2NF. El algoritmo para la evaluación de las claves, inicializa un conjunto H al vacio y un contador k a 0. Al finalizar el proceso H contendrá las claves y el contador el número de claves distintas. Este algoritmo consta de un bucle principal en las DF de la base. Para cada DF(i-sima) se comprueba si su parte izquierda PI_i esta contenida en H; si es así no se realiza ninguna operación, en caso contrario se hace un bucle secundario en las restantes DF, para cada DF(j-sima) se comprueba si la intersección de PI_i y PI_j es vacia en caso afirmativo no se hace operación alguna si no es así se reasigna a PI_i la unión de PI_i y PI_j. Al final de este bucle secundario se asigna a k el valor k+1 y a H_k el valor PI_i final, reasignandose a H la unión de H y H_k.

Al final se obtiene una colección de H_1, H_2, ...,H_k que serán las claves de las 2NF.

En función de los H_j j=1,2, ...,k se construyen k relaciones R_j en las que aparecen los H_j y los dominios B_s que dependan funcionalmente de H_j. En el supuesto de que estos B_s no dependan funcionalmente de ningún subconjunto de H_j distinto del total esta R_j está en 2NF. En el caso de que algún B_s dependa de un subconjunto SH_j de la clave H_j se elimina este B_s de R_j. Al eliminarlo se contempla la posibilidad de que en otra R_ℓ aparezca SH_j en cuyo caso se agrega alli el B_s, volviendo a considerarse la relación así obtenida para el estudio anterior. Si no existe ninguna relación de las creadas en las que aparezca SH_j se define una nueva relación cuyos dominios son SH_j y B_s.

Al final se comprueba si existe alguna relación R_j que sea unión de otras, eliminandose en este caso R_j.

ALGORITMO DE OBTENCION DE LA BASE DE DATOS EN 3NF.

Uno de los métodos mas simples de obtener relaciones a partir de un conjunto dado de dependencias funcionales, es agrupar en una misma relación todos los atributos que son funcionalmente dependientes de un mismo conjunto de atributos. Esto sugiere el siguiente procedimiento: Primero, dividir el conjunto de dependencias funcionales en grupos de tal forma que todas las DF en cada grupo tengan idénticas partes izquierdas. Segundo, para cada grupo construir una relación que conste de todos los atributos que aparecen en ese grupo. De esta forma, la parte izquierda de cada DF en el grupo es una clave de la relación correspondiente. Este método tiene, sin embargo, algunos inconvenientes:

- Las relaciones obtenidas no están en 3NF. Esto es debido a redundancias en el conjunto de DF.

- Las partes izquierdas de cada DF de un grupo, no son necesariamente claves de la relación formada a partir de esas DF; aunque siempre serán superclaves de la relación.

- Este método da origen en muchos casos a más relaciones de las necesarias.

- Al no considerarse las reglas de composición de las DF las DF redundantes que entren a formar parte de una relación, introducirán en ella atributos extra y contribuirán con ello a conexiones no normalizadas entre atributos.

Asi pues, una primera medida para aliviar el problema de la normalización, es el obtener una cobertura no redundante del conjunto de DF. Sin embargo, esta medida no es suficiente para evitar los problemas que las DF pueden causar en una relación; nos referimos al problema de los atributos innecesarios dentro de una dependencia funcional. Se dice que un atributo es innecesario si su eliminación no altera la clausura del conjunto de DF.

La eliminación de atributos innecesarios en las DF, ayuda a evitar dependencias parciales y superclaves que no son claves, en una relación R.

Una propiedad utilizada en el algoritmo para obtener la tercera forma normal es: Si dos relaciones tienen claves, que son funcionalmente dependientes la una de la otra (o sea, son equivalentes) entonces las dos relaciones pueden unirse. Según se vio antes la condición que debe cumplir una relación para que esté en 3NF es que ningún atributo no primario sea transitivamente dependiente de cada clave de la relación. Para probar que ningún atributo no primario es transitivamente dependiente de cualquier clave de una relación R, enunciamos la siguiente propiedad: Un atributo A se dice que satisface la propiedad P en la relación R, si verifica la siguiente proposiciónN Sea H una cobertura no redundante de un conjunto F de DF. Si la DF K -> A está en H y K -> A se utiliza para formar la relación R, entonces para cualquier atributo primario B de R, la DF K -> B puede ser derivada sin usar para ello la DF K -> A, o sea puede ser derivada de H-(K-> A). Se demuestra que una violación de la propiedad P puede inducir a una violación de la 3NF(5). Para obtener una relación R en 3NF, basta con eliminar de cada relación R todo atributo no primario que sea transitivamente dependiente de alguna clave de R, cambiándolo a otra o creando una nueva relación, con lo que el esquema de relaciones resultantes englobaría a las mismas DF.

Una vez expuestos todos los conceptos anteriores, pasamos a exponer el algoritmo en el que se ha basado la obtención de la 3NF (6).

PASO-1: Eliminar atributos innecesarios. Sea F el conjunto de

DF dado. Eliminar atributos innecesarios de la parte izquierda de cada DF en F, obteniendo como resultado el conjunto G.

PASO-2: Encontrar una cobertura (H) no redundante de G.

PASO-3: Partición. Dividir H en grupos, de tal forma que todas las DF pertenecientes a un grupo tengan idénticas partes izquierdas.

PASO-4: Mezclar claves equivalentes. Tomar J igual al conjunto vacio. Para cada par de grupos, H_i y H_j, con partes izquierdas X e Y, respectivamente, mezclar H_i y H_j si existe una biyección en H^+ entre X e Y. Para cada biyección, agregar $X \rightarrow Y$ e $Y \rightarrow X$ a J. Para cada A perteneciente a Y, si $X \rightarrow A$ está en H, entonces borrar esta DF de H. Hacer lo mismo para cada B perteneciente a X, tal que $Y \rightarrow B$ está en H.

PASO-5: Eliminar dependencias transitivas. Encontrar un H' contenido en H tal queÑ $(H' + J)^+ = (H + J)^+$, y ningún subconjunto propio de H' tiene esta propiedad. Agregar cada DF de J a su correspondiente grupo de H'.

PASO-6: Construir relaciones. Para cada grupo, contruir una relación que conste de todos los atributos que aparecen en ese grupo. Cada conjunto de atributos que aparecen en la parte izquierda de cualquier DF en el grupo, es una clave de la relación (el PASO-1 garantiza que ninguno de esos conjuntos contiene atributos extra). El conjunto de relaciones construidas constituyen un esquema para el conjunto de DF dado.

ALGORITMO DE OBTENCION DE LA BASE DE DATOS EN 4NF.

Una vez obtenida la BD en 3NF para el paso a 4NF se consideran las DMV que el diseñador de la BD introdujo. Para obtener la BD en 4NF inicializamos un conjunto R en donde se incluyen todas las 3NF. El algoritmo consta de un bucle general sobre todas las DMV; para la i-sima DMV, se considera un elemento de R que incluya la parte izquierda y derecha de la DMV, sea R_i; si allí la DMV es de tipo trivial no se modifica R, en caso contrario se evalua el número de atributos que dependen funcionalmente de la parte izquierda de la DMV, si ese número coincide con el grado de R_i se sigue con la siguiente DMV, si ese número es no nulo y distinto al grado de R se define una nueva relación R_i' que tiene por dominios la parte izquierda y todos los dominios que dependen funcionalmente de dicha parte izquierda de la DMV, si ese numero es cero se define una R_i' que consta únicamente de la partes izquierda y derecha de la DMV. Por último se modifica R_i eliminándole los atributos que dependen funcionalmente de la parte izquierda de la DMV considerada, agregando R_i' a R y sustituyendo la nueva R_i en R.

APENDICE

Consideremos una empresa textil, que posee datos referentes a sus proveedores, que se citan a continuación:

P#	Identificador del proveedor
NOM-P	Nombre del proveedor
CIUDAD	Ciudad de residencia del proveedor
REGION	Región de residencia del proveedor
C#	Identificador del componente
NOM-C	Nombre del componente
COLOR	Color del componente
CANT	Cantidad total pedida
P-M	Precio por metro
TIPO	Tipo de proveedor
BENEFICIO	Beneficio obtenido
AÑO	Año

Hagamos algunas hipótesis razonables sobre estos datos como pueden ser que todas las componentes independientemente del proveedor tienen un único precio por metro, y que un mismo proveedor puede tener dos tipos distintos (p.ej. sea proveedor de forros y telas).

Con estas hipótesis, y las características intrínsecas a los restantes datos, las dependencias funcionales y multivaluadas serán:

P# -> NOM-P
P# -> REGION
P# -> CIUDAD
CIUDAD -> REGION
P#.C# -> CANT
C# -> P-M
C# -> NOM-C
C# -> COLOR
P#.C# -> P-M
NOM-P.TIPO.AÑO -> BENEFICIO

NOM-P ->-> BENEFICIO.AÑO
NOM-P ->-> TIPO

Al introducir en el sistema esta colección de dependencias se obtiene:

INFORMACION SOBRE LOS ATRIBUTOS INNECESARIOS

EL ATRIBUTO P# ES INNECESARIO EN LA DEPENDENCIA FUNCIONAL: 9

INFORMACION SOBRE EL CALCULO DE
LAS DEPENDENCIAS FUNCIONALES BASICAS

LA DEPENDENCIA FUNCIONAL NUMEROÑ 2 , NO ES BASICA
LA DEPENDENCIA FUNCIONAL NUMEROÑ 6 , NO ES BASICA
EL RESTO DE LAS DEPENDENCIAS FORMAN UNA COBERTURA

BASE DE DATOS EN SEGUNDA FORMA NORMAL

LA BASE DE DATOS EN SEGUNDA FORMA NORMAL
ESTARIA FORMADA POR LAS SIGUIENTES RELACIONES :

* R01(P#,C#,CANT)
  CLAVE = P#,C#

* R02(CIUDAD,REGION)
  CLAVE = CIUDAD

* R03(NOM-P,TIPO,AÑO,BENEFICIO)
  CLAVE = NOM-P,TIPO,AÑO

* R04(P#,NOM-P,CIUDAD)
  CLAVE = P#

* R05(C#,NOM-C,COLOR,P-M)
  CLAVE = C#

BASE DE DATOS EN TERCERA FORMA NORMAL

LA BASE DE DATOS EN TERCERA FORMA NORMAL
ESTARIA FORMADA POR LAS SIGUIENTES RELACIONES:

* R01(P#,NOM-P,CIUDAD)
  CLAVE= P#

* R02(CIUDAD,REGION)
  CLAVE= CIUDAD

* R03(P#,C#,CANT)
  CLAVE= P#,C#

* R04(C#,NOM-C,COLOR,P-M)
  CLAVE= C#

* R05(NOM-P,TIPO,AÑO,BENEFICIO)
  CLAVE= NOM-P,TIPO,AÑO

BASE DE DATOS EN CUARTA FORMA NORMAL

LA BASE DE DATOS EN CUARTA FORMA NORMAL ESTARIA FORMADA POR LAS SIGUIENTES RELACIONES:

* R01 (P#, NOM-P, CIUDAD)
  CLAVE= P#

* R02 (CIUDAD, REGION)
  CLAVE= CIUDAD

* R03 (P#, C#, CANT)
  CLAVE= P#, C#

* R04 (C#, NOM-C, COLOR, P-M)
  CLAVE= C#

* R05 (NOM-P, AÑO, BENEFICIO)
  CLAVE= NOM-P, AÑO, BENEFICIO

* R06 (NOM-P, TIPO)
  CLAVE= NOM-P, TIPO

REFERENCIAS

(1). J. L. Becerril, R. Casajuana, F. Valer Sistemas de gestion de bases de datos relacionales, Inforprim 1978.

(2). E. F. Codd, A Relational Model of Data for Large Shared Data Banks. Comm. ACM Vol. 13, No 6, Junio 1970. pp. 377-387.

(3). C. J. Date, An Introduction to Data Base Systems. Addison-Wesley 1975.

(4). E. F. Codd, Further Normalization of the Data Base Relational Model. Courant Computer Science Symposia 6, Data Base Systems. Nueva York, 24-25 Mayo, 1971, pp. 33-64, Prentice-Hall, Nueva York.

(5). C. Beeri, P. A . Bernstein, Computational Problems Related to the Design of Normal Form Relational Schemas. ACM Trans. on Database Systems, Vol. 4,No. 1, March 1979.

(6). P. A . Bernstein, Synthesizing Third Normal Form Relations from Functional Dependencies. ACM Trans. on Database Systems, Vol. 1,No. 4, December 1976.

(7). R. Fagin, Multivalued Dependencies and a New Normal Form for Relational Databases. Research Report , IBM Research Laboratory, San Jose, California, Febrero 1977.

(8). H. K. Wong, N. C. Shu, An Approach to Relational Database Schema Design. Research Report RJ2688, IBM Research Laboratory, San Jose, California, Febrero 1980.

(9). W. W. Armstrong, Dependency structures of data base relationships. Information Processing 74,North-Holland Pub. Co., Amsterdam, 1974.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

DISEÑO DE UN SISTEMA DE ARCHIVOS DISTRIBUIDO PARA LA RED LOCAL UNIANDES

Edilberto Sánchez
Edgar Ruiz
Roberto Pardo

Departamento de Ingeniería de Sistemas
Universidad de Los Andes
Apartado Aéreo 4976, Bogotá - Colombia.

O B J E T I V O

El objetivo de este artículo es describir las principales decisiones de diseño de un sistema de archivos distribuído para la "Red Uniandes". En el diseño se integran una serie de soluciones a problemas relacionados con este tipo de sistemas. El artículo describe la arquitectura del sistema, estructura de nombres, directorio, lenguaje de trabajo y "software" (1) de control de concurrencia.

PALABRAS CLAVES: procesamiento distribuído, sistema de archivos distribuido, datos replicados, transacción, atomicidad.

(1) En el artículo se utilizan los términos del inglés "software", "hardware", "lock", "host", "hashing", "livelock","deadlock", "buffer", por no tener una traducción ampliamente utilizada en español.

1. INTRODUCCION

En este artículo se describen las principales decisiones de diseño de un sistema de archivos distribuído para la "Red Uniandes". Otros artículos (ARAU80) y (GAIT80), describen los aspectos de Software y Hardware de comunicación de la red.

La Red Uniandes es una red experimental -actualmente en fase de diseño- que nació como un esfuerzo conjunto entre los departamentos de Ingeniería Eléctrica y Sistemas de la Universidad de los Andes. Uno de los objetivos del proyecto es crear una infraestructura de bajo costo para desarrollo experimental de software distribuído. El esquema general de la red es el de un conjunto de nodos heterogéneos que a través de interfases se interconectan al compartir un medio de transmisión común.

Tradicionalmente los sistemas de archivos se han utilizado para manipular y almacenar información en una sóla máquina, es decir han sido centralizados. Aún bajo este esquema existen problemas de diseño interesantes como: decidir sobre la estructura de nombres y directorios, incluír mecanismos de confiabilidad, definir las primitivas de la interfase, etc.

Un posible esquema de funcionamiento de un sistema de archivos en una red es presentarle al usuario tantos sistemas de archivos (posiblemente diferentes) como nodos haya en la red. Este esquema sin embargo, es muy inflexible. El problema que se ataca en este artículo, es por el contrario, el de diseño de un sistema de archivos homogéneo a través de la red, donde tanto los programas para manipular archivos como los archivos mismos están repartidos en varios nodos, y es el sistema de archivos el que internamente maneja los problemas asociados con la dispersión.

En la literatura del tema se encuentran unas pocas propuestas de sistemas de archivos distribuídos y sobre todo gran cantidad de algoritmos para resolver problemas que pueden o no atacarse en este contexto, como control de concurrencia, atomicidad, confiabilidad, etc. Los objetivos principales del sistema de archivos distribuído aquí descrito son: 1) integrar en un sólo sistema una serie de soluciones a problemas varios relacionados con este tipo de sistemas, y 2) profundizar en detalles de soluciones a problemas que poco se mencionan al describir estos sitemas.

El artículo está dividido en ocho secciones. En las dos primeras secciones se aclaran algunos conceptos básicos y se mencionan algunos sistemas relacionados. Las otras seis secciones discuten el diseño del sistema de archivos para la red. La sección 4 discute el medio ambiente e introduce ejemplos para motivar el uso del sistema y para ilustrar sus funciones. La siguiente sección -la 5- describe el sistema propuesto partiendo de una arquitectura de niveles de donde se deducen los módulos internos (y sus relaciones) para la realización de las funciones. En la sección 6 se explican brevemente los mecanismos de control de concurrencia usados cuando se desea proveer atomicidad (confiabilidad) de transacciones y replicación de datos. Finalmente en la sección 7 se dan recomendaciones de implantación y en la sección 8 se mencionan algunas conclusiones. Más detalles sobre el diseño se pueden encontrar en (SANC80).

2. CONCEPTOS BASICOS

A continuación se definen una serie de términos y conceptos que se usarán a través del artículo.

Sistema de Archivos: Es el conjunto de Software del sistema operacional que se ocupa del manejo y organización tanto lógica como física de la información. Este sistema como mínimo crea un mecanismo para nombrar los archivos, genera y usa información sobre los archivos mismos (directorios), y provee ciertas primitivas para que los usuarios y/o aplicaciones manipulen los archivos. Adicionalmente el sistema puede soportar múltiples organizaciones de archivos, mecanismos de protección y confiabilidad, etc.

Sistema de Archivos Centralizado: Es un sistema de archivos en el que tanto el software para manejo de archivos como los archivos mismos están en un sólo nodo. Estos sistemas pueden ser o no ser usados en ambiente de red.

Sistema de Archivos Descentralizado: Es un sistema de archivos en el que la información puede estar ubicada en diferentes nodos y manipulada independientemente. Sin embargo, el usuario debe coordinar el acceso a cada uno de los sistemas de archivos locales que necesite. Es similar a un conjunto de sistemas de archivos centralizados repartidos sobre diferentes nodos de una red.

Sistema de Archivos Distribuído: Es un sistema de archivos que tiene las mismas características de un sistema de archivos descentralizado pero que realiza sus funciones por medio de la cooperación e interacción entre los diferentes sistemas de archivos en la red. Este sistema coordina automáticamente las referencias a diferentes nodos. Este tipo de sistema, con respecto al descentralizado, es más fácil de usar y evita el mal uso que el usuario podría hacer sobre él. Con respecto al centralizado este sistema puede ser más confiable, más eficiente (en términos de retrasos de comunicación y throughput) y más natural ya que la información puede estar cerca de donde se genera y usa.

Directorio: Es un archivo (o un conjunto de archivos) que contiene información sobre los archivos mismos. Por ejemplo contiene información sobre los dispositivos donde están los archivos, sobre su organización, sobre sus características físicas, etc. El sistema descrito en este artículo tiene un directorio que adicionalmente contiene información sobre los nodos de la red donde residen los archivos.

Acción: Es una llamada directa de acceso (lectura, escritura) a un archivo que cuando es local, o sea cuando se ejecuta sobre un archivo del mismo nodo, satisface condiciones de serialización y atomicidad.

Transacción: Es la unidad lógica de trabajo que consta de una o más acciones; es la llamada externa de un usuario al sistema de archivos. Si todas las acciones de la transacción son locales, la transacción es centralizada. Si una o más acciones de la transacción se refieren a archivos en diferentes nodos, la transacción es distribuída.

Propiedad de Atomicidad: Garantiza que todas las escrituras se hacen o ninguna se hace, independientemente de fallas. A nivel de acción, la atomicidad se refiere a acciones de escritura (o se hace o no se hace). A nivel de transacción, la atomicidad se refiere a un conjunto de acciones de escritura locales o remotas (o se hacen todas o ninguna se hace).

Propiedad de Consistencia: Garantiza que la concurrencia se efectúe correctamente. O sea, si se están ejecutando acciones intercaladas de diferentes transacciones (es decir transacciones concurrentes), cada transacción ve un panorama consistente de los datos como si las transacciones fueran ejecutadas una a la vez (serialmente). Cuando hay copias de archivos existen dos tipos de consistencia: a) Consistencia Interna: el resultado de acciones intercaladas es equivalente a un despacho serial de las transacciones, b) Consistencia Mutua: al suspender las actualizaciones las copias eventualmente convergen a valores idénticos.

Transacción Comprometida: Es un estado del progreso de ejecución de una transacción en el cual se decide que los efectos de su ejecución eventualmente se harán permanentes.

Propiedad de Confiabilidad: Garantiza que aunque ocurran fallas las transacciones comprometidas se lleven a cabo.

Propiedad de Recuperación: Garantiza que los efectos de las transacciones comprometidas se lleven a cabo y los de las otras (las no comprometidas) se deshagan luego de la recuperación de una falla.

3. TRABAJOS RELACIONADOS

En esta sección se describen algunos sistemas relacionados como RSEXEC - (THOM79) y WFS (SWIN79) .

RSEXEC (The Resource Sharing Executive)

Es un sistema de archivos distribuído que crea un medio ambiente el cual facilita compartir recursos de computación y almacenar información entre computadores ("host") de la red ARPA (DAVI79), de tal forma que los grandes computadores "aumentan" los recursos de los pequeños. Entre los principales comandos que provee se encuentran: WHERE y LINK que son usados para establecer diálogo en línea con otro usuario, comandos de configuración BIND, LIST y APPEND. (Por ejemplo: BIND (dispositivo) IMPRESORA (a nodo) SISTEMAS).

El modelo tomado por el RSEXEC mantiene información acerca de los archivos no locales más referenciados por un usuario y adquiere información de estos por medio de programas servidores remotos cuando es necesario. RSEXEC tiene una estructura de nombres de tipo jerárquico, por lo tanto para obtener una configuración determinada es necesario fijar los nombres de archivo, dispositivos y host los cuales son transformados en direcciones físicas. Esta función se realiza por medio de programas servidores. El usuario tiene la facilitad de manejar los archivos en forma compacta debido a las características de los comandos. No tiene problemas con el olvido de nombres de los archivos ya que los puede ubicar en su "contorno".

WFS (A Simple Shared File System for a Distributed Environment)

Es un servicio de archivos compartidos que opera en un medio ambiente local de computación distribuída implantado en ETHERNET (METC76). El sistema provee un conjunto conciso de operaciones, entre las cuales hay operaciones para :a) leer y escribir sobre páginas de archivos; b) asignar y liberar páginas de archivos; c) obtener y modificar propiedades de archi-

vos y d) realizar actividades de mantenimiento al sistema. Las operaciones que más se usan son las de lectura y escritura de páginas, las cuales pasan como parámetro la identificación del archivo (FID) y el número de página; por ejemplo: Readpage (fid, pagenum). Las operaciones están diseñadas de tal forma que si se repite una acción antes de confirmarla, tendrá el mismo efecto que una sola.

El directorio de WFS es implantado como una tabla de "hashing" de tamaño fijo, contigua y en una dirección de disco conocida; adicionalmente está duplicado en los servidores los cuales son computadores con gran capacidad de almacenamiento. La estructura de nombres es de tipo planar compuesta de: a) identificadores de los archivos que existen en la red; b) páginas de transformación de identificador lógico a dirección física de cada archivo, y c) archivos divididos en páginas de tamaño fijo. Esta estructura facilita la adición de un mecanismo de recuperación y de control de transacciones debido a la fragmentación del archivo en páginas.

4. MEDIO AMBIENTE

SAD (se usa esta abreviación en adelante para referenciar el sistema propuesto) utiliza las facilidades que le provee el protocolo para comunicación entre procesos para establecer canales virtuales entre dos procesos remotos por medio del cual se envían mensajes en forma confiable (GAIT80). El protocolo de transferencia de archivos permite expandir operaciones sobre un archivo local a nivel de red, como crear, borrar, transferir, etc.

El usuario del sistema de archivos distribuído invoca las transacciones que se encuentran almacenadas en una librería de transacciones. Otro tipo de usuario más familiarizado con el sistema elabora las transacciones mismas ya sea mediante un lenguaje específico o haciendo uso directo de las primitivas que le provee el sistema.

El siguiente ejemplo ilustra el uso de transacciones y algunos problemas que debe solucionar SAD

```
/* Se otorga un préstamo en base al valor e ingresos */
/* del empleado                                      */

Trans: PRESTAMO_A_EMPLEADO (#_EMPLEADO, VALOR, OTORGADO)
       P1: Lea sueldo, bonificaciones, deducciones de
           #_EMPLEADO.
           Calcule ingresos
           if ingresos >= 20% de VALOR then   OTORGADO = true
                                      else    OTORGADO = false.

end trans

/* Se elabora un transpaso de fondos cumpliendo con */
/* una condición de mínimo ingreso predeterminada   */

Trans: ABONO_A_FONDO_DE_EMPLEADOS (#_EMPLEADO, CANTIDAD)
       D1:  Lea ingresos de # EMPLEADO. Retire CANTIDAD a #_EMPLEADO
```

```
D2: Lea ingresos de fondo.
    Abone CANTIDAD a ingresos de fondo.
    if ingresos de #_EMPLEADO< mínimo_ingreso
       then escriba viejo_ingreso de #_EMPLEADO
                    viejo_ingreso de fondo.
end trans
```

En el * ejemplo se supone que los archivos usados se encuentran en nodos diferentes al nodo donde se lleva a cabo la transacción. Cuando las transacciones se ejecutan se pueden presentar varios problemas. Por ejemplo, al ejecutarlas en forma serial si ocurre una falla en el nodo donde están los items del fondo de empleados luego de ejecutar la acción D1, se haría un retiro inválido al empleado al no realizarse D2. Al ejecutarlas en forma concurrente si P1 se efectúa después de D1 y el resultado de la acción D3 es dejar los viejos valores, la transacción Préstamo_A_Empleado ejecuta con un dato falso. El prevenir este tipo de problemas es un objetivo de SAD.

5. DESCRIPCION DEL SISTEMA

5.1 Características

Las principales características de SAD son:

-Los archivos están divididos en páginas para facilitar el manejo de buffers, apuntadores y registros (relativos a páginas).
-Los servidores coordinan las transacciones del nodo, dan respuesta a los pedidos externos de otros servidores y se encargan de la interfase con el sistema de archivos local.
-Las operaciones sobre los archivos son agrupadas en transacciones.
-El sistema provee un conjunto definido de primitivas adecuadas para la creación de transacciones.
-Existe un mecanismo de control de concurrencia por medio de "locks" a nivel de página y archivo. Los locks tienen asociado un tiempo de inactividad sobre estos items; cuando el tiempo de inactividad se completa, automáticamente se deshacen los locks hechos por la transacción.
-Se provee un mecanismo para mantener consistencia entre las copias remotas de los archivos replicados.
-Los mecanismos de control entre servidores garantizan las propiedades de consistencia, atomicidad, confiabilidad y recuperación.

5.2 Arquitectura

El sistema está estructurado en cuatro niveles. El nivel 0 provee las facilidades básicas para comunicación entre procesos, transferencia de archivos y la interfase con el sistema de archivo local. El nivel 1 borra los límites físicos en cuanto al uso de los archivos, controlando concurrencia, manteniendo atomicidad y asegurando confiabilidad. El nivel 2 corresponde al despachador de transacciones que por medio de la interfase (primitivas) con el nivel 1, coloca en forma ejecutable las transacciones en el contexto de los mecanismos de control. El nivel 3 comprende la aplica-

ción y la interfase con el usuario. Para una mayor comprensión de la arquitectura el lector puede referirse a la figura 1.

Nivel 3

Existen dos tipos de usuarios del sistema: unos como por ejemplo el cajero del Fondo de Empleados que invoca la transacción ABONO_A_FONDO_DE_EMPLEADOS o el jefe de este Fondo que invoca la transacción PRESTAMO_A_EMPLEADO (ver sección 4), quienes solo conocen su efecto o sea solo invocan transacciones; y otros, más familiarizados con el sistema y con la aplicación, quienes elaboran las transacciones.

Nivel 2

Este nivel está conformado por los siguientes módulos: a) Configura Directorio de la Transacción. Provee al usuario que crea la transacción un medio para conocer los archivos disponibles para la elaboración de la transacción y crea un directorio lógico cuando se crean instancias de las transacciones en base a los parámetros provistos por quienes hacen uso de la transacción. b)Librería de Transacciones. Usado para almacenar las transacciones elaboradas, guardando el directorio y las acciones de la transacción (escritas en un lenguaje específico), como P1 y P2 de PRESTAMO_A_EMPLEADO. c) Contexto de la Instancia. Al recibir el pedido de ejecución por parte del usuario, se encarga de crear una instancia de la transacción en base a los parámetros pasados y asociar un descriptor de la instancia que contiene además de otra información un apuntador al directorio lógico de la instancia, conformado por el módulo 2. Extrae las acciones de la transacción de la librería y las pasa al módulo traductor con los cambios involucrados por los parámetros dados por el usuario. d) Traductor, se encarga de traducir a primitivas del sistema las acciones de las transacciones pasadas por el módulo anterior, haciendo uso del directorio lógico de la instancia.

Nivel 1

Este nivel corresponde a los mecanismos de control y se encarga de resolver los problemas de concurrencia, atomicidad (confiabilidad) de transacciones y consistencia de datos replicados cuando se ejecutan transacciones concurrentemente en un medio de computación distribuído.

Se hace uso de archivos de datos e intenciones, que están divididos en páginas. Un descriptor (mapa de páginas) se utiliza para cambiar los datos del archivo. El archivo de intenciones se usa para completar las transacciones suspendidas cuando ha habido una falla. Adicionalmente provee un mecanismo de control de inactividad sobre los items, asociando a cada uno un tiempo máximo. Las anteriores funciones las realiza el servidor de cada nodo, el cual coordina las acciones de las primitivas y los pedidos externos de los servidores remotos.

Nivel 0

Provee las facilidades básicas para comunicación entre procesos, transferencia de archivos y la interfase con el sistema de archivos local. Estas facilidades son operadas por un conjunto definido de primitivas que permiten

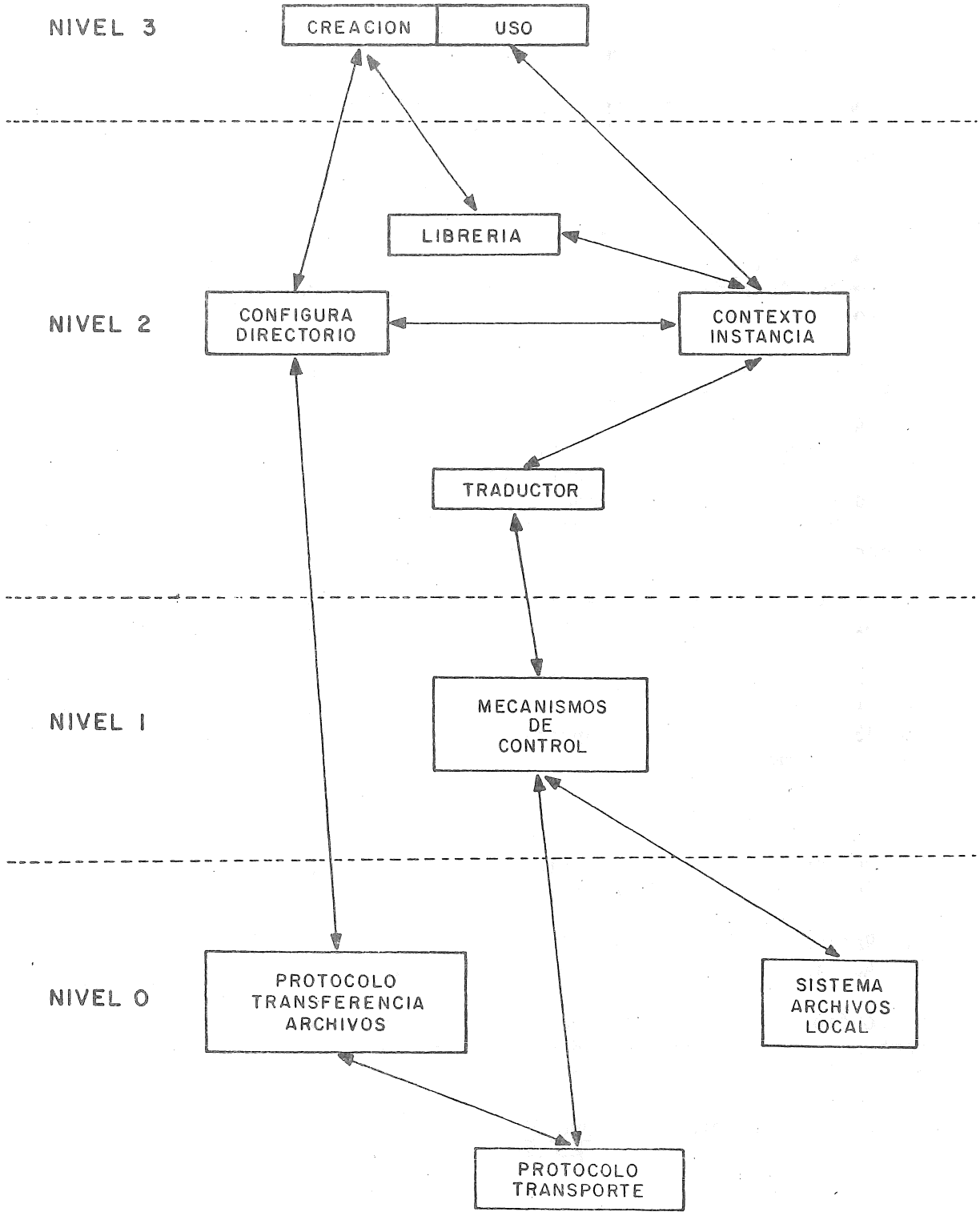


FIGURA 1. ARQUITECTURA DE SAD

establecer comunicación con procesos remotos, intercambiar mensajes confiablemente, transferir archivos en una forma segura y hacer acceso a la información local.

5.3 Estructura de Nombres

Un problema clásico de diseño ya sea en Sistemas de Archivos Centralizados, descentralizados o distribuídos es decidir cual es la estructura de nombres más adecuada, de tal manera que se pueda manipular fácilmente y opere en forma conveniente.

La estructura de nombres de SAD es de tipo jerárquico basada en la expansión de la estructura de árbol que proveen algunos sistemas operacionales. Esta estructura fue escogida para darle modularidad al sistema y permitir la expandibilidad.

La estructura se divide en cuatro niveles de la siguiente forma: a) En el primer nivel se identifican los nodos (computadores) de la red que han sido ligados por medio de la configuración establecida para la transacción. b) En el segundo nivel se identifican los dispositivos de cada computador. c) En el tercer nivel se identifican los archivos que se encuentran en cada dispositivo seleccionado. d) En el cuarto nivel se identifican los mapas de páginas de los archivos de datos que transforman páginas lógicas en direcciones físicas de disco e identifican páginas sin asignar.

La figura 2 describe claramente este tipo de estructura. Un ejemplo que ilustra el funcionamiento de la estructura de nombres es el siguiente:

Un usuario identificado con el código 001 por medio de la transacción Trans: TRASPASO-ESTUDIANTE cambia de facultad al estudiante cuya identificación es 7611432, de Sistemas de Economía. Para ello invoca la transacción en la siguiente forma:

Trans : TRASPASO-ESTUDIANTE (001, 7611432, SISTEMAS, ECONOMIA)

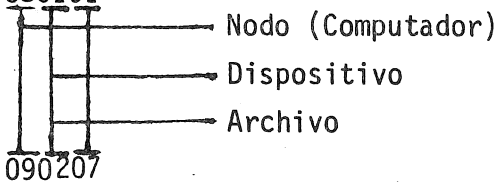
Al crearse la instancia de esta transacción las acciones quedan conformadas sintácticamente de la siguiente manera:

A1: Retire (SISTEMAS, ESTUDIANTES, 7611432)
A2: Adicione (ECONOMIA, ESTUDIANTES, 7611432)

El configurador de contexto en este punto opera en base a la estructura de nombres para conformar la identificación de los archivos, tomando los datos del directorio de la instancia así:

Sistemas.Estudiantes = 080101

Economía.Estudiantes = 090207



Como se observa hasta ahora se tienen tres niveles de la estructura de nombres: computador, dispositivo y archivo. La página no se ha ubicado debido a que falta establecer si el archivo es local o remoto. Sí se supone que la transacción se coordina en el nodo de la facultad de sistemas, por medio de los mecanismos de control se obtiene que la primera acción se debe ejecutar localmente mientras que para la segunda se debe realizar un pedido a un nodo remoto (Facultad de Economía); luego para la primera acción se tendría definido toda la estructura de nombres al invocarse el sistema de archivos local por medio de la primitiva correspondiente, mientras que la estructura de la segunda es completada por el sistema de archivos del nodo remoto.

5.4 Directorio

Otro problema interesante de diseño es el de directorio. En un sistema distribuído el directorio debe tener una localización transparente para el usuario, debe recopilar toda la información necesario para llevar a cabo las transacciones operadas sobre la red, y debe ser eficiente (tener un buen tiempo de respuesta) y tener un mecanismo sencillo de mantenimiento.

Teniendo en cuenta que el medio en que se va a implantar este sistema consta de pocos nodos y tratando de cumplir con las condiciones anteriormente expuestas, se unifican los directorios de cada nodo conformando un directorio global el cual contiene información sobre: los nodos (Computadores) los dispositivos de almacenamiento y los archivos en cada unidad.

También el directorio global contiene algunas características de los archivos, como por ejemplo, si están replicados o no, el código de protección, etc. Este directorio es replicado en los nodos que puedan mantenerlo ya sea por capacidad disponibilidad o por decisiones administrativas. Aquellos nodos que por una u otra razón no tienen un replicado del directorio global, tienen una tabla que les permite ubicar cuáles nodos cercanos lo contienen.

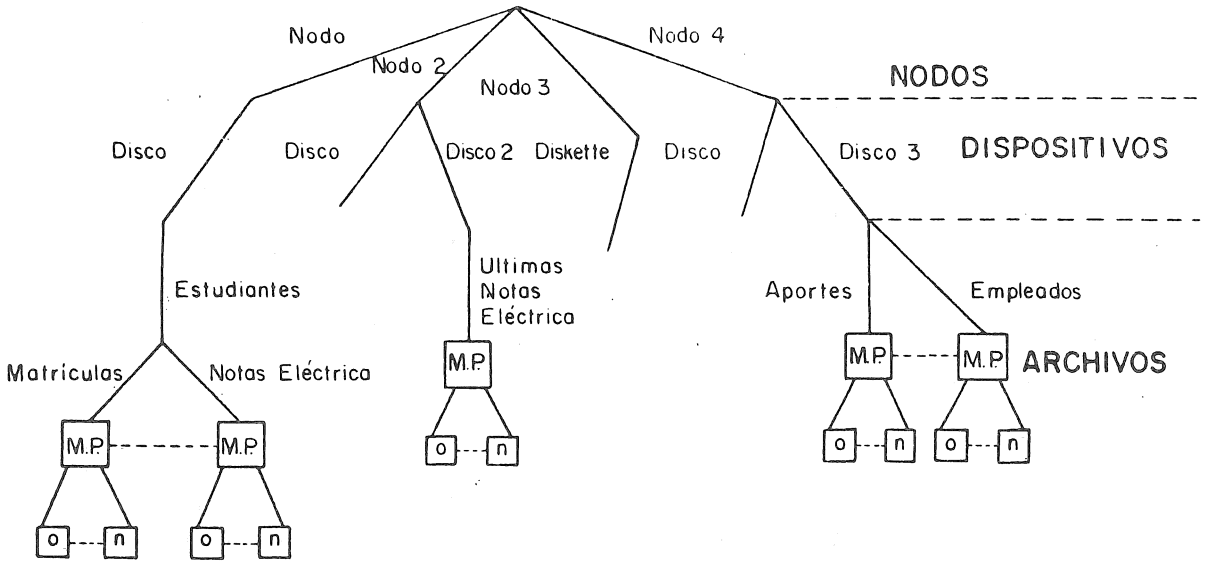
Como se observa en la figura 3 el directorio es de tipo jerárquico e implementado como listas con apuntadores. Este directorio es usado por el sistema en los siguientes casos: a) Cuando un usuario lo necesite para elaborar una nueva transacción, b) Cuando se va a crear el directorio lógico para la instancia de una transacción que se carga desde librería y cuyos archivos de trabajo son asociados de acuerdo a los parámetros del usuario.

Los siguientes ejemplos ilustran la diferencia entre el uso estático y dinámico de los archivos por medio del módulo Configura Directorio de Transacción.

Ejemplo 1: Supongamos que los empleados de una entidad están registrados con todos sus datos en un sólo archivo, este sería fijo en el directorio lógico de una transacción como:

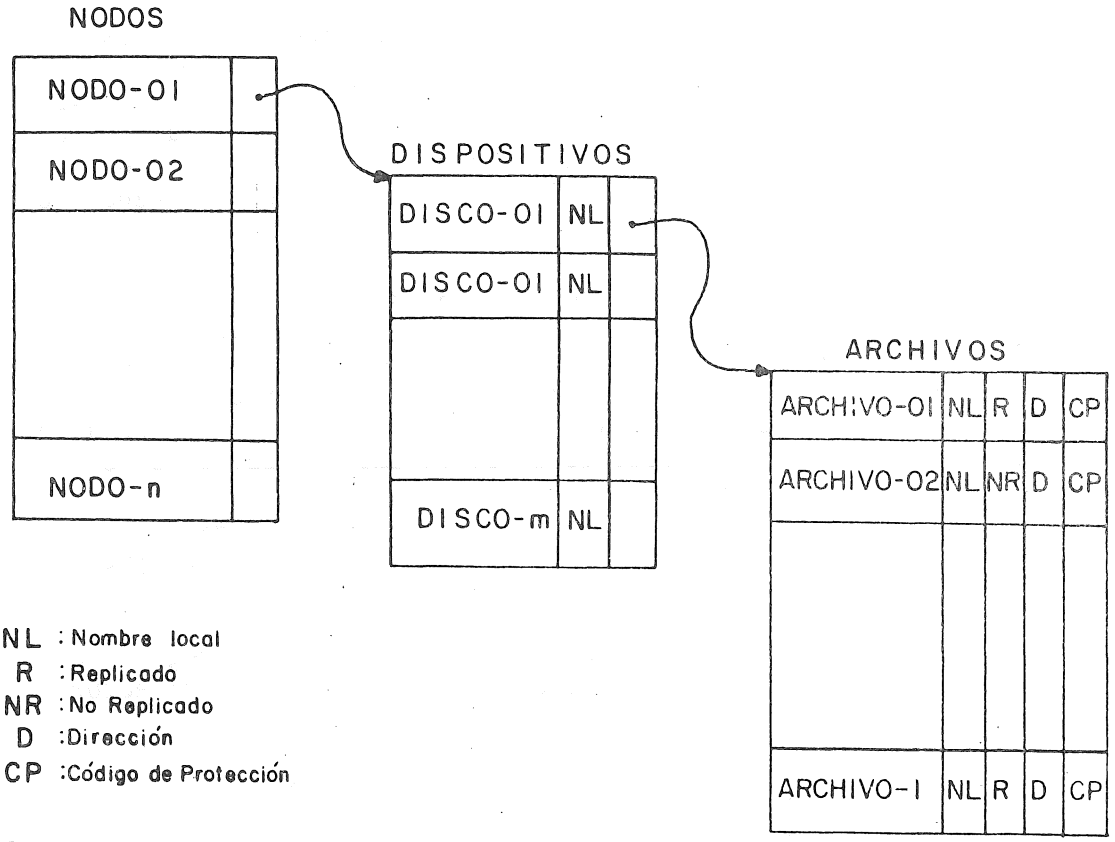
Trans: ABONO_A_FONDO_EMPLEADOS (#_EMPLEADO, CANTIDAD)

El descriptor asociado a esta transacción sería:



M.P Mapa de Páginas

Fig. 2. Estructura de Nombres de SAD



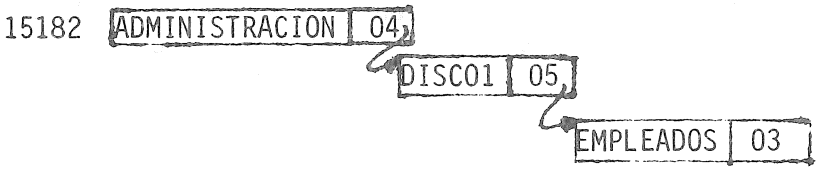
NL : Nombre local
R : Replicado
NR : No Replicado
D : Dirección
CP : Código de Protección

Fig. : 3 Componentes del Directorio Global de SAD

DESCRIPTOR

Nombre: ABONO_FONDO_EMPLEADOS
Número Transacción: _____
Prioridad Externa : 10
Código de Protección: 05
Entrada al Directorio: 15182

DIRECTORIO LOGICO ABONO_A_FONDO_EMPLEADOS



Ejemplo 2: Supongamos que en la Universidad cada facultad tiene un archivo de estudiantes, donde cada facultad corresponde a un nodo de la red. En una transacción de traspaso de estudiante los archivos pueden variar según el caso así:

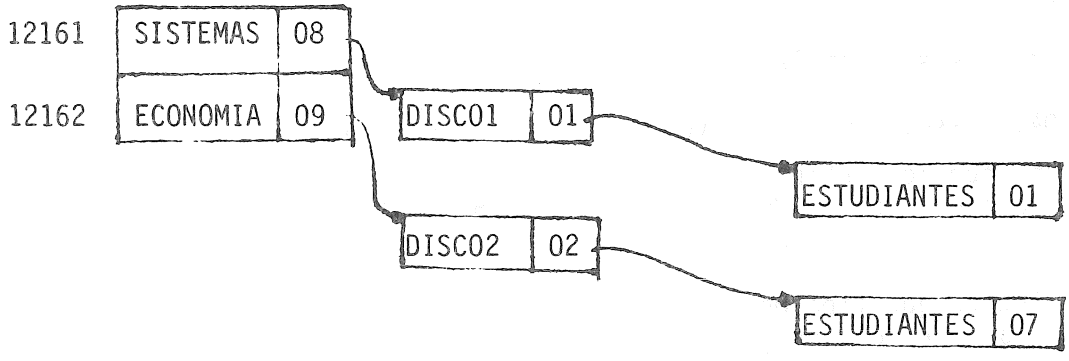
Trans: TRASPASO_ESTUDIANTE (001, 7611432, SISTEMAS, ECONOMIA)

El descriptor asociado a esta transacción sería:

DESCRIPTOR

Nombre: TRASPASO-ESTUDIANTE
Número Transacción: _____
Prioridad Externa: 15
Código de Protección: 01
Entrada al Directorio: 12161

DIRECTORIO LOGICO TRASPASO-ESTUDIANTE



El valor del número de la transacción se coloca cuando los mecanismos de control empiezan a operar sobre la transacción traducida. En el primer ejemplo el directorio lógico por ser estático es cargado directamente de librería, en tanto que, en el segundo ejemplo es conformado mediante la interacción con alguno de los nodos que contiene el directorio global.

5.5. Lenguaje de Trabajo

La interfase entre el nivel 2 y el nivel 1 se realiza por medio de un conjunto definido de primitivas, que para su operación envuelven un intercambio de mensajes entre servidores cuando el archivo referenciado es remoto, y una comunicación directa entre el servidor y el sistema de archivos local cuando recibe un pedido bien sea externo o interno a un archivo del nodo.

Para estandarizar el acceso a los archivos de la red se establece un identificador de archivo (IDA), siendo este un número entero compuesto de la concatenación del número del nodo, número de unidad y número de archivo en la unidad. Su longitud en bits depende de la cantidad de aquellos en la red.

Primitivas

Algunas de estas primitivas son:

LEA PAGINA (IDA, tipo, número, área, desplazamiento, #_transacción).

Esta primitiva tiene como función permitir el acceso a una página de un archivo. Los parámetros indican lo siguiente:

Tipo: Indica si es la llave de un registro o el número de una página

Número: Corresponde al valor de la llave o número de la página.

Area: Identifica el número de buffer donde se almacena la página leída. Si el Tipo es una llave se trae toda la página donde se encuentra el registro referenciado.

Desplazamiento: Localiza el registro dentro de la página.

#_Transacción. Identifica el número de transacción respecto al nodo en que se usa la primitiva.

ESCRIBA PAGINA (IDA, número, área, #_transacción)

Tiene como función escribir una página de un archivo.

Los parámetros son similares a los de LEA-PAGINA.

LOCK (IDA, clase, número, llave_acceso, #_transacción)

Tiene como función inhibir el acceso a otras transacciones hasta que no se realiceuna operación de UNLOCK o que se deshaga el LOCK por inactividad. Los parámetros indican lo siguiente:

Clase. Indica qué tipo de lock se va a realizar: si es a nivel de página o a nivel de archivo, y si va a ser lectura o escritura.

Número. Es un número de llave o página por medio del cual se identifica la página a inhibir.

Llave_Acceso. Es retornadà por el servidor para ser usada en operaciones subsecuentes (al utilizar las primitivas LEA o ESCRIBA).

UNLOCK (IDA, clase, llave_acceso, número #_transacción)

La descripción de los parámetros es similar a los de LOCK. Su función es desbloquear el Item para permitir el acceso a otras transacciones que lo necesiten.

ASIGNE PAGINA (IDA, #_ página)

Su función es retornar el número de una página libre identificada en el mapa de páginas .

RETIRE PAGINA (IDA, #_página)

Tiene como función colocar en la lista de libres la página especificada.

Se ilustra el uso de algunas de estas primitivas con el ejemplo de la transacción TRASPASO-ESTUDIANTE, El nódulo traductor forma un programa compuesto de instrucciones en lenguaje de trabajo y primitivas del sistema, partiendo de la instancia de la transacción que le provee el módulo que configura el contexto.
Así, la transacción lista para ejecutar es:

```
EMPIECE TRANSACCION
LOCK (080101,PAGINA,7611432, ____,_____)
LOCK (090207,PAGINA,7611432, ____,_____)
LEA-PAGINA(080101,PAGINA,7611432,BUFFER3,____)
ESCRIBA-PAGINA(080101,____,BUFER,_____)
LEA-PAGINA(090207,PAGINA,7611432,BUFER5,_____, _____)
        :
ESCRIBA-PAGINA(090207, _____,BUFER5,_____)
        :
UNLOCK(080101,PAGINA,____,____,_____)
        :
FIN TRANSACCION
```

Los parámetros que aparecen con guión (_____) toman su valor en la fase de ejecución cuando entran a operar los mecanismos de control. Los puntos se usan para indicar las operaciones de la transacción que no son traducidas a primitivas del sistema por no tener relación directa con el acceso a la información.

6. MECANISMOS DE CONTROL DE CONCURRENCIA

A continuación se describe en prosa un resúmen del algoritmo básico que implanta los mecanismos de control de concurrencia cuando hay replicación y se desea garantizar atomicidad (confiabilidad) de transacciones. La descripción algoritmica aparece en (SANC80).

6.1 Caracteristicas

Existen dos problemas que se presentan potencialmente en un medio ambiente donde se ejecutan procesos concurrentemente, estos son: "livelock", el cual

consiste en un número ilimitado de intentos para lograr un lock sobre un archivo o página del archivo; y "deadlock", el cual corresponde a una función en la que cada una de las transacciones de un conjunto S de dos o más transacciones están esperando hacer un lock sobre una página o archivo inhibido por alguna otra transacción en el conjunto S (ULLM80).

Una estrategia FIFO (first-input-first-output) elimina livelocks. Gran variedad de soluciones han sido propuestas por los diseñadores de sistemas operacionales para resolver problemas de deadlock (BRIN73), (DIJK68), por lo tanto no se discute en el artículo para hacer énfasis en los mecanismos de control de concurrencia citados anteriormente.

Facilidades de los Nodos

Se supone que los nodos proveen acceso aleatorio a un sólo sector (página) a la vez. El sistema propuesto maneja en una forma lógica los locks a nivel de página elaborados por las transacciones, liberándolos ya sea por mandato explícito o implícitamente en caso de sobrepasar un tiempo determinado de inactividad sobre la página o archivo.

Tipos de Archivos.

Se tienen dos tipos de archivos: archivos de datos y archivos de intenciones. Los archivos de datos contienen la información operada por las transacciones, están estructurados de tal forma que una modificación lo convierte en una nueva versión que puede tener muchas páginas cambiadas.

Se provee un nivel de transformación entre las páginas lógicas y las páginas físicas. La primera página física de cada archivo de datos es un descriptor del archivo que contiene un mapa de páginas indicando la página física donde la página lógica está almacenada. Adicional al mapa de páginas, existe una lista de páginas reales libres y una variable por cada página que indica la transacción, si hay alguna, en la que esté envuelta el archivo o página.

Los archivos de intenciones tienen una variable de estado, un número de transacción, una lista de cambios y una secuencia de los descriptores de los archivos modificados por la transacción. La transacción puede estar en alguno de los tres estados siguientes: iniciada, comprometida o completa. Cuando la transacción se encuentra en su estado de inicio, se la puede abortar; cuando está en el estado comprometida, eventualmente todas las escrituras serán realizadas aún en presencia de fallas hasta que llegue al estado completa. La lista de cambios contiene identificadores de los archivos de datos que han sido modificados en la transacción y apuntadores a copias de sus nuevos descriptores. Este archivo de intenciones está localizado en el nodo que coordina la transacción. Esta estructura es similar a la usada por -- (LAMP79).

6.2 Descripción

A continuación se describen las operaciones más relevantes que se ejecutan en el contexto de los mecanismos de control: Inicie Transacción, lock, lea,

escriba, aborte transacción y fin transacción. Cada una de estas operaciones básicas lleva asociada un procedimiento"Inicie Transacción" asigna un archivo de intenciones y retorna un número de transacción que es pasado como parámetro por los demás procedimientos.

Para visualizar mejor el algoritmo se describe primero suponiendo que no existen replicados como el algoritmo descrito por Paxton (PAXT79) y luego se explican los cambios necesarios.

Cuando el nodo que está coordinando la transacción contiene el archivo referenciado ejecuta directamente la acción de la primitiva, en caso contrario hace un pedido al programa servidor del nodo donde se encuentre el archivo referenciado.

INICIE TRANSACCION

Asigna un archivo de intenciones a la transacción y retorna un número de transacción relativo al nodo, el cual es pasado como parámetro adicional para los demás procedimientos . Este número sirve para identificar las primitivas de las diferentes transacciones cuyas invocaciones pueden venir entre mezcladas (concurrencia).

LOCK

Recibe el IDA y retorna la llave de acceso al archivo para ser usada en los accesos siguientes. Para ello estableci si el archivo es local o remoto, Si es local consulta su directorio y mira si la página o archivo está envuelto en alguna transacción. En tal caso la nueva transacción debe esperar y generar un nuevo pedido hasta que complete un número de intentos preestablecidos. En caso de ser un archivo remoto , en base a la primera componente del IDA se toma el número del nodo el cual se utiliza para establecer comunicación con el servidor de dicho nodo, por medio del Protocolo de Comunicación entre Procesos (PCP). Establecida la comunicación se ensambla un mensaje con el tipo de pedido requerido. El servidor remoto al recibir el pedido de lock verifica si el item referenciado está comprometido en alguna transacción (incompleta por una falla en ese nodo), en cuyo caso invoca el procedimiento de recuperación el cual se encarga de completar la transacción anterior. Cuando se otorga el lock sobre el item se retorna las propiedades de este. Al expirar el tiempo de inactividad sobre un item, automáticamente se deshacen los locks que la transacción había hecho sobre otros items, y se pasa al estado de la transacción a completa sólo si no estaba comprometida.

LEA

En base a las propiedades del item retornadas por el "lock" se verifica si el registro identificado por el parámetro (número) de la primitiva se encuentra en una página leída anteriormente y se retorna el desplazamiento dentro de la página. En caso contrario se genera un pedido de página bien sea el archivo local o remoto, transfiriéndose una nueva página.

ESCRIBA

En base al buffer especificado por la primitiva, se transfiere la página bien

sea al servidor local o remoto dependiendo del IDA; el servidor por medio de la primitiva Asigne_Página localiza una página libre donde se realiza la escritura, luego retorna el número de la página física donde quedó la página transferida para que el coordinador de la transacción actualice el mapa de páginas que le había sido transferido al invocar la primitiva lock.

ABORTE TRANSACCION

Los locks que había hecho la transacción sobre los items de trabajo son liberados, así como las estructuras de datos necesarias para la ejecución de la instancia de la transacción.

FIN TRANSACCION

Dependiendo del número de archivos modificados existen tres casos:

-Si no se modifica ningún archivo (lo cual se detecta mediante un bit de cambio en el descriptor local del archivo en la instancia), se liberan los archivos inhibidos y se cambia el estado de la transacción a completa. Otra posibilidad es dejar que los locks se rompan por tiempo de inactividad.
-Si se modifica un sólo archivo, se liberan los de sólo lectura, se cambia el descriptor del archivo remotor por el descriptor local (enviando la lista de páginas lógicas cambiadas) y se pasa el estado de la transacción a completa.
-Si se modifican más archivos se debe ir creando el archivo de intenciones para prevenir una falla. Se siguen los siguientes pasos:

1- Se liberan todos los items (páginas y archivos) de solo lectura y los que no se modificaron (para ello se utiliza el bit de cambio a nivel de página y archivo).
2- Se hace un pedido de lock local al archivo de intenciones.
3- Se marca cada item modificado con el número de la transacción y el IDA del respectivo archivo de intenciones para la instancia de la transacción.
4- Se copia en el archivo de intenciones la lista de cambios de cada archivo modificado, y se pasa el estado de la transacción a comprometida. El estado de la transacción se puede pasar a comprometida debido a que no hubo problemas con otras transacciones y el servidor remoto permitió marcar el archivo.
5-Se reescribe el nuevo descriptor del archivo (enviando la lista de cambios) y se retira la marca indicando que el item ya no está envuelto en ninguna transacción, luego se libera el lock sobre el item.
6- Se cambia el estado de la transacción a completa y se libera el lock del archivo de intenciones.

Procedimiento de Recuperación

Es invocado cuando un item referenciado por la primitiva Lock está involucrado en una transacción (marcado) o se rompe el lock del archivo por inactividad.

Al invocarse como resultado de la primitiva Lock, se libera el lock del item y se hace un pedido de lock al archivo de intenciones, para luego hacer un

nuevo pedido de lock al archivo de datos. Esto evita posibles deadlocks cuando un procedimiento tiene inhibido los items y otro el archivo de intenciones. Cuando se logra tener el archivo de intenciones sólo se libera hasta que se completa la transacción, aún esperando a que se liberen los locks de los items. Al invocarse como resultado de la inactividad sobre el item su primera acción es tratar de hacer un lock al archivo de intenciones. El archivo de intenciones se utiliza para ver el estado en que quedó la transacción antes de la falla y proceder a completarla en caso necesario.

Tratamiento para Replicados

Como se enunció anteriormente es necesario hacer algunos cambios para describir el algoritmo cuando existen replicados. En la ejecución de una transacción se opera con una de las copias disponible, efectuando los cambios requeridos en las páginas libres. En el procedimiento Fin_Transacción que es cuando se forma el archivo de intenciones, el servidor que coordina la transacción hace un pedido al servidor encargado del archivo para modificarlo, en caso de que existan replicados el servidor encargado espera confirmación de los otros nodos para efectuar los cambios requeridos dándose la posibilidad de abortar la transacción cuando exista conflicto con otras transacciones. Este algoritmo para actualizar copias es similar al descrito por Ellis. - (ELLI77).

Para la toma de decisión los servidores pueden estar en uno de estos estados: primero, que el servidor al que se le esté pidiendo el voto no haya llegado a la fase de formar el archivo de intenciones de la transacción que esté coordinando y segundo, que esté en la fase de formar el archivo de intenciones. La decisión en el primer estado en caso de conflicto es simple puesto que la transacción que hizo el pedido está más adelantada, abortándose la otra. En el segundo estado cuando ambos están formando los archivos de intenciones, se sugieren las siguientes alternativas de decisión en caso de conflicto: a) Se decide por el que haya recolectado un número mayor de confirmaciones según los archivos replicados utilizados (una transacción puede modificar varios archivos y algunos de estos estar replicados) pudiendo efectuar un promedio ponderado de los archivos marcados y no marcados (en esta fase todos los archivos que van a ser modificados quedan marcados); al hacerse el pedido se enviará información sobre los archivos ya marcados; b) Por registros de tiempo (cuenta eventos), para dar prioridad a la más antigua. Para el sistema se escogió la primera debido a que es más sencilla de implantar y provee más información actualizada. Se tiene en cuenta que el pedido para la actualización de un replicado conlleva un tiempo de inactividad sobre el archivo mucho mayor que el normal, debido a que el procedimiento para recolectar "listos" puede ser demorado, además se va formando el archivo de intenciones primero con los que no estén replicados para obtener un mejor rendimiento. Estos cambios se tienen también en cuenta en el procedimiento de recuperación.

7. NOTAS DE IMPLANTACION

En esta sección se describe un modelo para la implantación SAD. El modelo está formado por una colección de programas servidores que corren en algunos nodos de la red escogidos a priori.

Servidor

Para el soporte de programas modulares (transacciones) se utiliza una construcción llamada servidor. El servidor está compuesto de objetos (enteros, arreglos, colas, archivos, páginas de archivos, procedimientos) y procesos. Los procesos de un servidor pueden compartir objetos, comunicarse con otros procesos del mismo servidor por medio de objetos compartidos y comunicarse con procesos en otros servidores únicamente por envio de mensajes. Durante la ejecución de una trasacción la población de servidores serán creados y servidores existentes pueden autodestruirse cuando no tengan más funciones que realizar. Un servidor se crea como consecuencia de un EMPIECE_TRANSACCION o de un pedido hecho por un servidor remoto (Protocolo para iniciar conexión). El servidor es responsable de su espacio direcccionable, el manejo de almacenamiento y los procesos de recuperación que son iniciados después de una falla (descritos en la sección 6). Ver figura 4. La estructura del servidor utilizada en este modelo es similar a la usada para guardianes por Barbara Liskov (LISK79).

Organización Servidor

La comunicación del servidor con su medio ambiente es por medio de pedidos generados en base a las primitivas del sistema. Como respuesta a un pedido crea un proceso para manejarlo (Figura 5).

El proceso creado se sincroniza con los otros por medio de un MONITOR para asegurar que sólo uno manipule los datos compartidos (table de locks, cola de mensajes, tabla de estado de los servidores, directorio) (HOAR74).

Procesos

Una de las funciones más importantes del servidor es satisfacer los pedidos de las transacciones invocadas por el usuario. Estos pueden ser satisfechos directamente o por medio de la interacción con los servidores remotos. El servidor lleva a cabo sus funciones por medio de los siguientes procesos:

1- El Proceso ESER (Estado del Servidor) se encarga de mantener la información de estado acerca de los programas servidores remotos. Este proceso mantiene actualizados los registros de estado de los programas servidores de otros nodos por intercambio de información de los procesos ESER de cada nodo. Cada proceso ESER reliza periódicamente un "broadcast" de su propia información de estado.
2- Por cada pedido (interno o externo) que tiene asociada la ejecución de una primitiva se genera un proceso PRIM que es el encargado de llevarla a cabo. Los procesos de este tipo fueron descritos en la sección 6, donde las estructuras de datos como archivos de intenciones y de datos son compartidos.
3- Los seudo-operandos EMPIECE-TRANSACCION, ABORTE-TRANSACCION YFIN-TRANSACCION general procesos de control de las transacciones como se describió en la sección anterior.

Monitor

El monitor es el encargado de: sincronizar la ejecución de los procesos que conforman parte del servidor, mantener actualizada la tabla de locks (a ni-

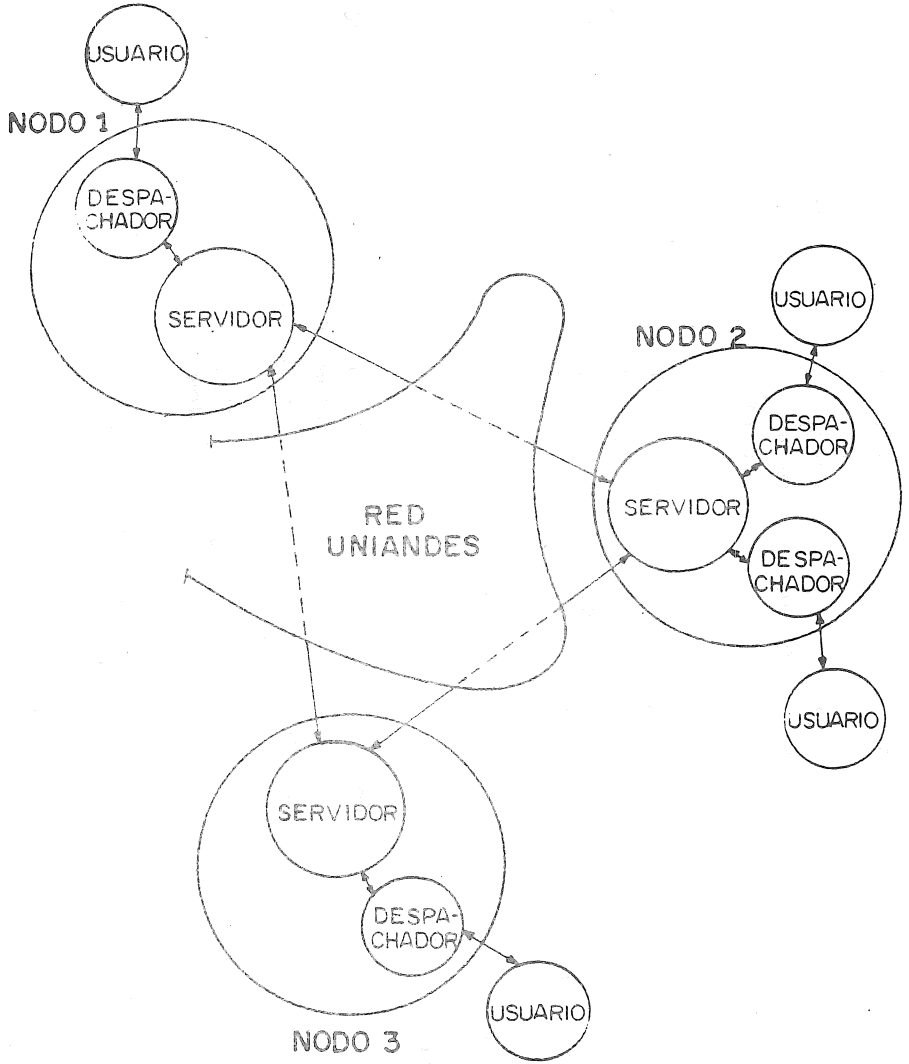
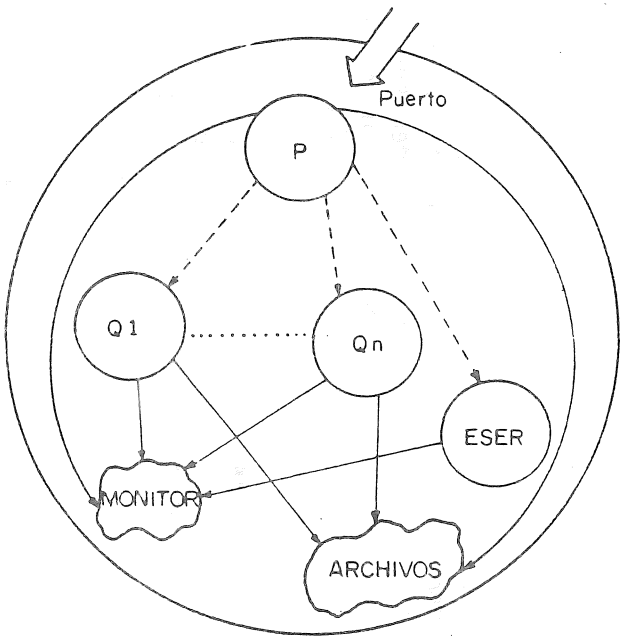


Fig. : 4. Estructura de Operación de SAD



P : Recibo de Pedidos

Qj : Proceso PRIMó Generado por Pseudo - operando

ESER : Estado - Servidores

Fig.: 5. Estructura Interna del Servidor

vel de página y archivo), manipular la cola de mensajes de los procesos en ejecución, asociar time-outs a cada mensaje y generar el proceso de recuperación.

Puertos y Mensajes

Un puerto es un punto de entrada a un servidor. Los puertos son las únicas entidades que tienen nombres globales para que cada proceso maneje directamente sus mensajes; los nombres de los puertos pueden ser enviados en mensajes, esto implica que se pueden recibir mensajes en un mismo puerto desde diferentes fuentes. La asignación dinámica entre buffer's y puertos es realizada por el protocolo de transporte.

Un mensaje consiste de un comando identificador y los argumentos necesarios. En el envio de mensajes para pedir un servicio el comando identificador corresponde al nombre de una operación que ha de ser invocada.

8. CONCLUSIONES

El diseño del sistema de archivos distribuído (SAD) para la Red Uniandes refleja las facilidades fundamentales que se requieren para dar soporte a una aplicación distribuída. La arquitectura de SAD abstrae niveles independientes lo cual reduce la complejidad y permite expandibilidad.

En el diseño de un sistema de archivos distribuído la arquitectura de nombres, directorio y primitivas deben ser bien definidas. La noción de transacción es fundamental para el control de concurrencia. Este diseño puede servir de base para la construcción de un sistema más sofisticado como una base de datos distribuída en la cual unproblema interesante de diseño es el modelo externo.

Para la implantación de un sistema de archivos distribuído se ve la necesidad de una herramienta que permita procesos integrados, monitores, variables de condición y otras facilidades, como los sitemas de lenguajes de Programación: PLITS (FELD79), MESA (MITC78) que son de considerable potencia y elegancia.

REFERENCIAS

(ARAU80) Araujo, L. y Fernándes, J. , "Diseño de la Interfase para la Red Local Uniandes", publicaciones del Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes. 1980.

(BRIN73) Brinch Hansen, P., Operating System Principles Prentice Hall, Inc. Englewood Cliffs, New Jersey, 1973.

(DAVI79) Davies, D., et. al., Computer Networks and their Protocols, John Wiley & Sons, 1979.

(DIJK68) Dijkstra, E.W., "Cooperating Sequential Processes", Programing Languages. (F. Genuys, ed.), Academic Press, N.Y., 1968

(ELLI77) Ellis, C.A., "A Robust Algorithm for Updating Duplicated Databases", Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks,

(FELD79) Feldman, J.A., "High Level Programing for Distributed Computing", CACM, Vol 22, 1979

GAIT80) Gaitán, L. y Paredes. R., "Diseño de Protocolos de Alto Nivel para la Red Local Uniandes", Publicaciones del Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes, 1980.

(HOAR74) Hoare, C.A., "Monitors: An Operating System Structuring Concept". CACM. Vol. 17, 1974. pp. 594-557.

(LAMP79) Lampson. B. y Sturgis, H., "Crash Recovery in a Distributed Data Storage System", Xerox PARC, Marzo 1979 (aparecerá en CACM)

(LISK79) Liskov, B. "Primitives for Distributed Computing", Proc. of 7th Symposium on Operating System Principles Diciembre 1979, pp. 33-42.

(METC76) Metcalfe, R. y Boggs, D., "Ethernet: Distributed Packet Switching for Local Computer Networks". CACM Vol. 21, 1976. pp. 395-403.

(MITC78) Mitchel, J.G., et. al., Mesa Language Manual, CSL Repor 79-3, Xerox PARC, Febrero, 1978.

(PAXT79) Paxton, W. , "Client-Based Transactiosn to Maintain Data Integrity" Proc. of 7th Symposium on Operating System Principles, 1979

(SANC80) Sánchez, E., Ruíz E., "Diseño de un sistema de Archivos Distribuído para la Red Local Uniandes", Proyecto de Grado, Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes, 1980.

(SWIN79) Swinchart, G., et. al., "WFS: A Simple Centralized File System For a Distributed Environment", unpublished paper Xerox Palo Alto Reserarch Center 1979.

(THOM79) Thomas, R.H., "A resource sharing executive for the ARPANET", National Computer Conference, 1973

(ULLM80) Ullman, J.D., Principles of Database System, Computer Science Press, 1980

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

A RELATIONAL DATA BASE MANAGEMENT SYSTEM

Ricardo O. Giovannone

DATA S.A.
Bdo. de Irigoyen 560
Buenos Aires - Argentina

RESUMEN

Un DBMS Relacional para la educación

Como resultado del proyecto "A Relational Data Base System", se construyó un programa llamado MINI DBMS1. Este programa, escrito en el lenguage de programación PASCAL (UCSD PASCAL), simula la actividad de un administrador de base de datos. El modelo de datos utilizado fue el relacional y el programa fue implementado en una Minicomputadora.

El objetivo principal de proyecto fue la implementación de un sistema que simule la actividad desarrollada en un DBMS, teniendose en cuenta fundamentalmente las distintas actividades que se desarrollan en la generación, mantenimiento e interacción con una base de datos a través de un DBMS. Se tuvo en cuenta que los usuarios serian estudiantes de un curso de base de datos y que tendrian que interactuar con el sistema y obtener como resultado el conocimiento conceptual del funcionamiento de un DBMS. Se asumió que los estudiantes o usuarios del sistema debian conocer el modelo relacional y las operaciones relacionales correspondientes.

Las operaciones del DBMS que se implementaron son: aquellas que realiza el administrador de la base de datos,(creación de las relaciones y destrucción de las mismas), aquellas operaciones para el mantenimiento de relaciones,(altas de tuplas,

bajas y cambias) y por último, las operaciones relacionales, proyección (projection), restricción (restriction), selección (selection), producto cartesiano (cross product) y acoplamiento (join).

El programa fue implementado en una minicomputadora, explotando las caracteristicas interactivas del lenguage y del hardware disponible (CRT, unidad central y unidad de diskette).

En el reporte correspondiente a este proyecto se cubren los siguientes puntos:

- Estructuras de datos utilizadas en el programa.
- Explicación conceptual del funcionamiento de los módulos que realizan las actividades principales del DBMS.
- Conclución.

Se quiere dejar claro que el objetivo de este sistema es educativo y no para el uso en aplicaciones comerciales. Esto se manifiesta en la implementación de la mayoria de las operaciones de un DBMS puesto que se considera una cantidad reducida de tuplas por relación en la base de datos.

1. INTRODUCTION

The result of the project "A Relational Data Base System" is a program called Mini DBMS1. This program has as its main objective the implementation of a data base management system in a minicomputer environment for educational purposes rather than applications-oriented software. The relational data model was the one used in this project.

The educational objective of the system concerns the exercise of data base administrator,relations maintenance, and relational operations activities utilizing a small number of relations and tuples. The data base administrator operations are creation and deletion of relations. The relations maintenance type of operations that can be performed are of common type to a data base. They are insertion of tuples or records, deletion, and update. The relational operations that can be exercised are projection, selection, restriction, cross product, and join. This assumes that the user of the system should have knowledge about the relational model and operations.

This project was divided into phases. Phase one concerned relational operations over one relation, and phase two involved relational operations combining two relations. The host language used in this implementation was UCSD PASCAL

version I. 4. This language allows the possibility of having machine-independent programs, which means that the programs developed in this language can run in a variety of minicomputers and microcomputers. The machines used at the moment are a Terak minicomputer and a PDP11/10. The project direction was under Prof. D. Dearholt, and the students involved were Ying-Ying Peng in phase one, and Ricardo Giovannone in phases one and two. This paper covers the work done in phases one and two of the project. The paper will cover explanations of program modules and strategies used.

2. DATA BASE MANAGEMENT SYSTEM STRUCTURE

The Mini DBMS1 is a set of program modules (procedures and functions) that make possible the handling of relations to and from the diskette. Relations are stored in the diskette and fetched to memory for further processing using basic procedures or functions. When the relational data are brought to memory, these are stored temporarily in either of two data structures (arrays) that have the same structure as the file records holding the relations in the diskette. These data structures serve as a working area for most of the procedures or functions that perform basic activities like retrieving,

printing, or recording. These arrays are also used in all the specialized procedures that perform the relational operations. They are also used by the procedures that do insertion, updating, and deletion of tuples in relations already created in the data base.

Figure 1 shows this conceptual view of the Mini DBMS1 activity. Figure 2 shows all the commands available in this Mini DBMS1 to exersice all the operations.

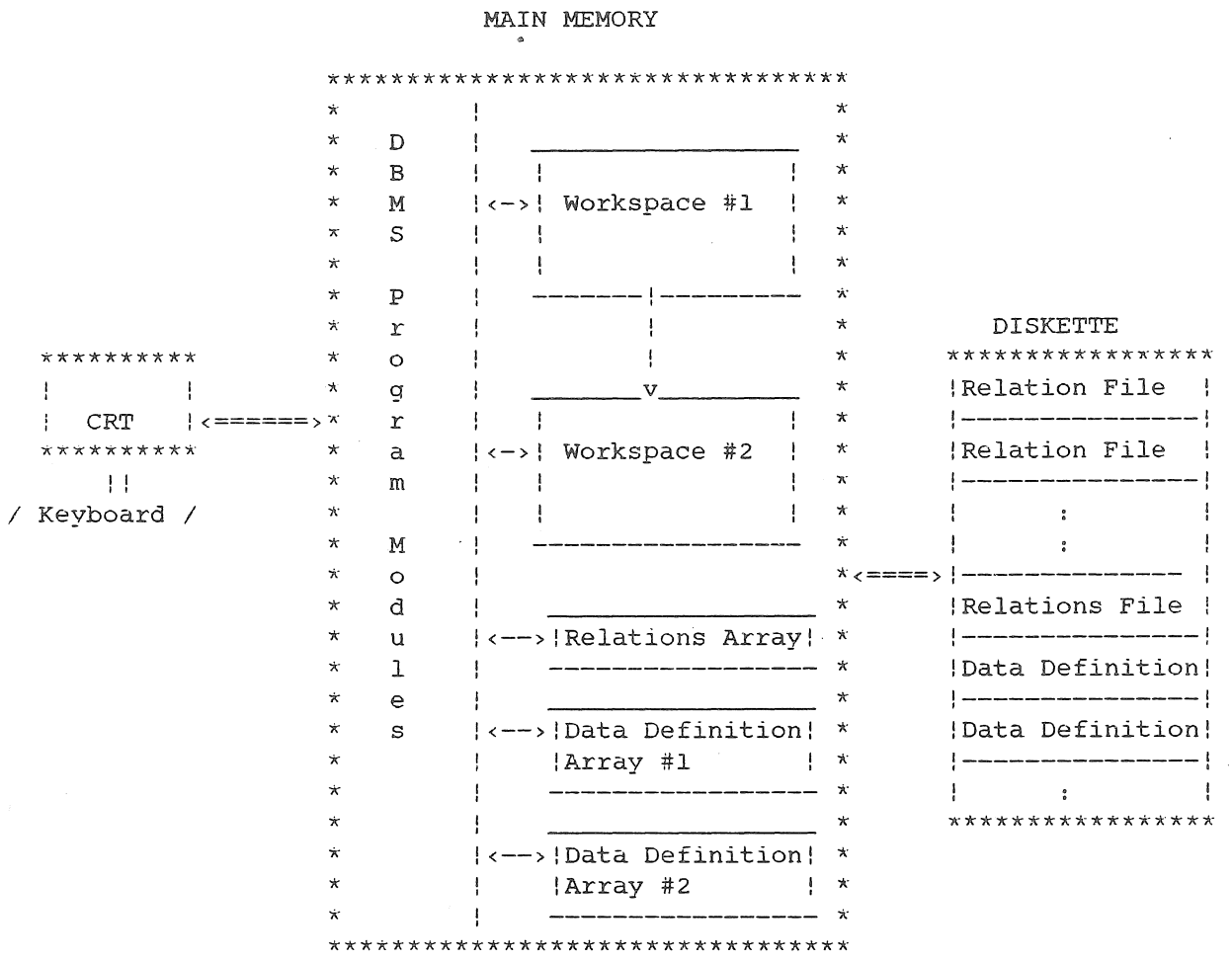


Figure 1. Conceptual view of the data base management activity.

To start the execution of the system Type: X , then a question like this will appear, What file?, and you Type: DBMS1

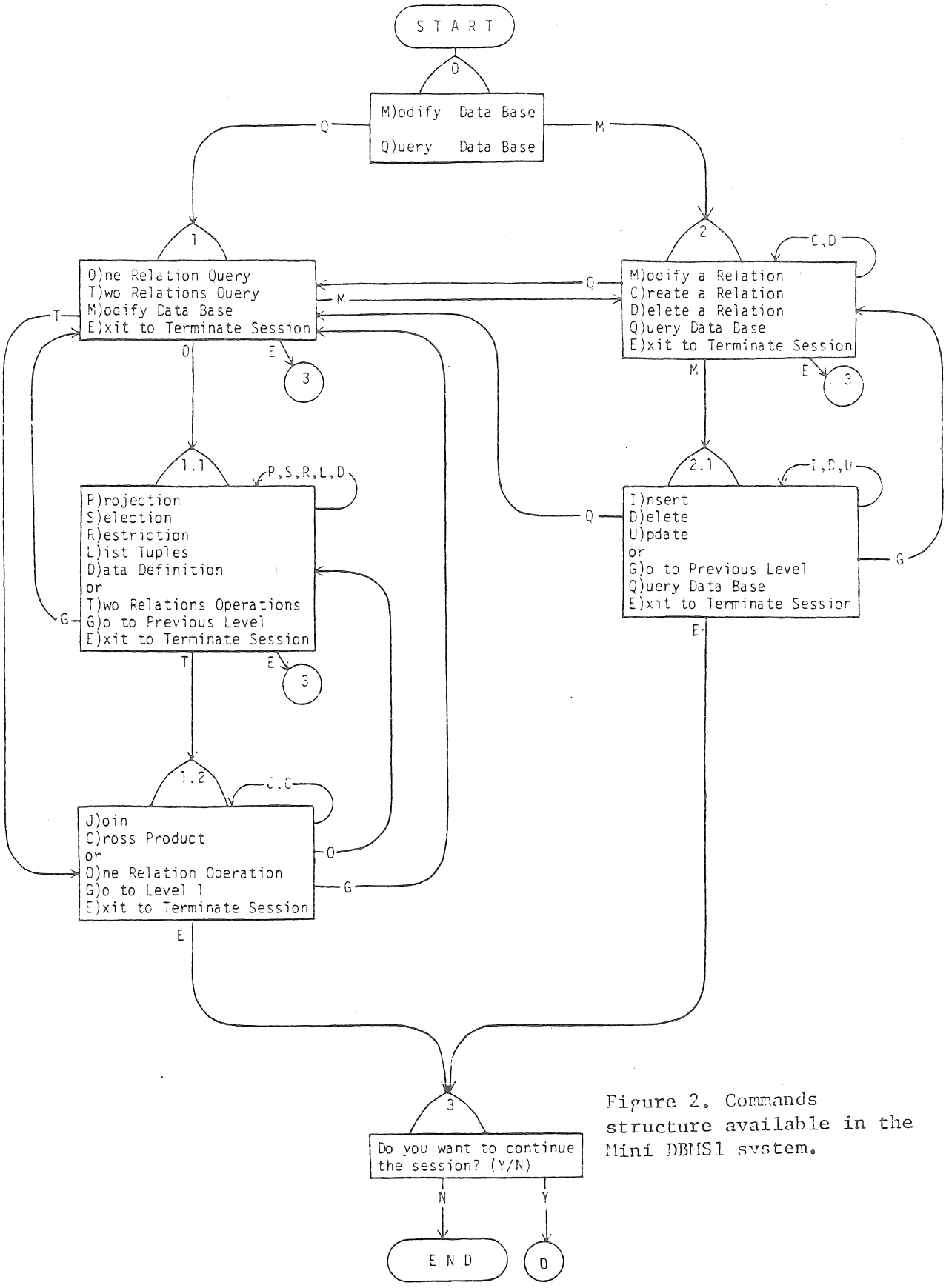


Figure 2. Commands structure available in the Mini DBMS1 system.

The following section will describe the different type of data structures that are generated and used by this Mini DBMS1.

2.1. Relation File

This file is composed of records of characters (maximum 50 characters). Each record holds one tuple of the relation. One of the constraints in using this structure is that not all the relations have tuples 50 characters long, so some space is wasted. The number of tuples that a relation can have is not a restriction. However, there is a restriction in terms of the working space created by the program in the main memory. The working area holds a maximum of 10 tuples per relation. Of course this can be changed just by changing the dimension of the working area, but one should have a memory bigger than 56K bytes.

2.2. Data Definition File

The data stored in this file refers to the different qualifications related to each attribute in a particular relation. The file name is associated to the relation name in

order to fetch it when needed. The information about each attribute is stored in a record. There is one record per attribute. The information concerns domain of the attribute, length, starting position in the tuple, designation of key or common attribute, and value ranges. There is one Data Definition file for each relation created.

2.3. Relations File

The purpose of this file is to store information about the current relations in the data base. There is a record for each relation. This record is formed by the fields relation name, number of tuples, and number of attributes.

The file structures presented are the only ones at the moment used by the program modules. The relation file and the data definition files are used for error checking purposes and to get parameters of information used in all the procedures and functions.

The relation file serves only as a means to store the relational data to be managed by the system.

3. PROGRAM MODULES

This section will refer to the different procedures or

functions used to execute data base administartor operations, relational operations and insertion, deletion, and updating of tuples.

3.1. Structure of Data Base

This command is produced automatically each time the user starts a session. The purpose of this procedure is to show the whole structure of the data base. This means that all the relation names currently in the data base and the corresponding data definition for each one will be shown. Figure 3 shows in a conceptual diagram the step followed by the procedure to accomplish this part.

3.2. Data Base Administrator Operations

3.2.1. Creation of a New Relation

This command allows the user to create a new relation for the data base. This task is always performed by the data administrator. At the moment this command has a security pass in order to prohibit access to unauthorized users. Figure 4 describes the steps involved in this command.

3.2.2. Deletion of a Relation

The delete command delete completely a relation from the data base. A special password is required of the user to perform this activity, which again should be performed by the data base administrator. The delete command appears under the M)odify Data Base command. This command erases the data file for the relation, the record from the relations file and the corresponding data definition file. Figure 5 describes this command.

3.3. Relational Operations

Relational operations are divided into one relation operations and two relations operations. The one relation operations involve projection, selection, and restriction. The two relations operations are cross product and join.

3.3.1. Projection

This relational operation is performed by following several steps. First, the relation desired is brought to main memory

and loaded to one of the working areas. The data in this working area is used as row data for a procedure that does the projection. The projection procedure selects the specified attribute in each tuple. These attribute values are transferred to the second working area. Before the transfer is done, a redundancy checking procedure over the correspondent attribute value is carried out, if the attribute value already exists in the workspace #2 then no transfer is done. Finally, all the attribute values selected are in the workspace #2. The user has the option of having this resulting projection sorted or not. This implementation of the projection operation is limited to the projection of one attribute values at a time. Figure 6 shows the steps for this operation.

3.3.2. Selection

The first part of this command is carried out like a projection. The relation is loaded into the first working area and then these data are taken to do the relational operation under the desired options and to pass the result to workspace #2 where the results of every operation are always stored. Figure 7 points out the different steps involved in this operation.

3.3.3. Restriction

This relational operation involves one attribute name and a corresponding domain value related by a relational operator. The first part of this operation is performed the same as the previous ones and the result is also placed in workspace # 2, and then printed out. Figure 8 represents the steps followed in this command.

```
LOAD Relations File
WHILE not end of Relations File DO
Begin
  PRINT Relation name
  LOAD Relational Data Definition
  PRINT Data Definition
End
```

Figure 3. Procedure that displays the data base structure.

```
LOAD Relations File
PRINT Relation Names in the Data Base
ASK for password to create Relations
IF correct password THEN
Begin
  ASK for the name for the new Relation
  IF New Relation Name non-redundant THEN
  Begin
    Data Definition construction
    Insert first tuples for the new Relation
  End
  ELSE access denied
```

Figure 4. Steps involved in the creation of a new relation.

```
ASK for appropiate password
IF password correct THEN
Begin
  ASK for relation name to be deleted
  PURGE corresponding Relation File
  PURGE corresponding Data Definition File
  DELETE relation name from Relations File
End
ELSE delete denied
```

Figure 5. Delete relation steps.

```
LOAD Desired Relation into Workspace   1
ASK which attribute is to be projected
WHILE not end of Relation DO
Begin
  COPY attribute value selected
  IF attribute value copied exists in workspace
       2 then NO TRANSFER
  ELSE TRANSFER value to Workspace   2
End
IF sorting option true THEN
Begin
  SORT contents of Workspace   2
  PRINT Workspace   2 content
End
```

Figure 6. Steps involved in projection operation.

```
LOAD Desired relation into Workspace   1
LOAD Data Definition into Data Def. array 1
ASK for first attribute name, relational operator,
    and second attribute name
WHILE not  end of relation DO
Begin
  IF current tuple matches desired qualification THEN
  Begin
    TRANSFER current tuple to Workspace   2
  End
End
PRINT Workspace   2 content
```

Figure 7. Procedure followed to perform selection operation.

```
LOAD Desired relation into Workspace   1
LOAD Data Definition into Data Def. array 1
ASK for first attribute name, relational aperator,
    and attribute value
WHILE not end of relation DO
Begin
  IF current tuple matches desired qualification THEN
  Begin
    TRANSFER current tuple to Workspace   2
  End
End
PRINT Workspace   2 content
```

Figure 8. Procedure followed to perform restriction operation.

```
ASK for first Relation name
LOAD Relation into Workspace   1
LOAD Data Definition into Data Def. array 1
PRINT Data Definition for this relation
ASK for second Relation name
LOAD Relation into Workspace   2
LOAD Data Definition into Data Def. array 2
PRINT Data Definition for this relation
WHILE not end of First Relation DO
Begin
  WHILE not end of Second Relation DO
  Begin
    CONCATENATE to tuple from first Relation tuple
                from second Relation
    PRINT result of concatenation
  End
End
```

Figure 9. Cross product steps.

3.3.4. Cross Product

This relational operation involves two relations from the Data Base. The tuples from one relation are related to a second relation. Each tuple from the first relation is related to all the tuples in the second relation, in this way generating new tuples that are the concatenation of tuples from the first and second relation. Figure 9 shows the steps followed in this command.

3.3.5. Join

The relational operation Join also involves two relations from the Data Base. The tuples from the first relation are related to the second relation tuples by a selection type of relational operation. This operation is done taking the attributes values from one tuple (from Relation 1) and compared with the other tuple (from Relation 2). Whenever there is a success in the comparison, based on the operator chosen (=, <>, >=, etc.) a new tuple is generated. This new tuple is obtained from the concatenation of the tuples from both relations. Figure 10 represents the conceptual steps involved in obtaining the join of two relations.

3.4. Relations Maintenance

3.4.1. Insert

This commands allows the user to insert new tuples to an already created relation. The relation is loaded into the two workspaces at the same time. The insertion takes place in Workspace # 2. The tuple is inserted at the next available space on the workspace. Then the whole relation with the insertion is shown to the user. The user has the alternative of keeping this insertion or starting all over again. After user approval the relation is sorted by the key and saved into the diskette. Figure 11 shows the steps in this activity.

3.4.2. Delete

The delete command refers to tuple deletion. This is done by loading the relation into the two workspaces. The user should enter the qualification where the deletion should take place (attribute name, relational operator, and attribute value). If the user wants to delete one tuple, then the value should be the one corresponding to the desired tuple. The other way to delete more than one tuple would be to use the qualification (attribute name, relational operator, and attribute name). In

this case the attributes should be compatible.

The procedure does an exchange of the tuple attribute values by the word "TUPLE DELETED. " After searching the whole workspace # 2, the resulting modification is shown to the user, with the word "tuple deleted" where deletion took place. The user has the alternative of accepting or rejecting this result. After accepting the resulted information, tuples without the word "TUPLE DELETED" are put back into the relation file. Figure 12 describes the steps involved in this command.

3.4.3. Update

Update of a tuple is done in the following way. The user should enter the key value for the relation to be updated. This is done by means of the qualification (attribute name, relational operator, and key value). Then the system will search for this tuple in workspace # 2. If the tuple is found, then the system will show the whole relation content where the old tuple exists. Then the user has the chance to update the whole or some of the attribute values in this tuple. Following

the update of the tuple, the system again shows the relation with the tuple updated. The user has the chance to approve this modification or reject it. If the user accepts the changes made, the relation will be saved. Figure 13 describes the steps for this command.

```
ASK for first Relation name
LOAD Relation into Workspace   1
LOAD Data Definition into Data Def. array 1
PRINT Data Definition for this relation
ASK for second Relation name
LOAD Relation into Workspace   2
LOAD Data Definition into Data Def. array 2
PRINT Data Definition for this relation
ASK for first attribute name from First Relation,
    relational operator, and
    second attribute name from Second Relation
WHILE not end of First Relation DO
Begin
  WHILE not end of Second Relation DO
  Begin
    GET attribute values from First Relation and Second
        Relation
    IF values match desired qualification THEN
    Begin
      CONCATENATE to Tuple from first Relation all
      attribute values of tuple from second Relation
    PRINT result of concatenation
    End
  End
End
```

Figure 10. Join operation steps.

```
LOAD Relation for insertion into Workspaces   1 &   2
LOAD Data Definition
WHILE not end insertion of tuples DO
Begin
  PROMPT lines for insertion
  ADD new tuple at the end of Workspace   2
End
SORT tuples in workspace   2 by the key
PRINT Relation with new tuples
IF affirmative answer to keep tuples THEN
Begin
  UPDATE number of tuples in Relations File
  UPDATE old Relation file in diskette
End
ELSE no insertion made
```

Figure 11. Steps involved in the insertion of tuples.

```
ASK for Relation Name
LOAD Relation into Workspaces   1 &   2
LOAD Data Definition into array 1
ASK for type of qualification to access tuple(s)
WHILE not end of relation DO
Begin
  IF tuple matches qualification THEN
  Begin
    REPLACE tuple for "TUPLE DELETED" string
  End
PRINT content Workspace   2
IF desired deletion THEN
Begin
  UPDATE number of tuples in Relations File
  UPDATE old Relation file in diskette
End
ELSE no deletion made
```

Figure 12. Steps to delete tuple(s) from a relation.

```
ASK for Relation name
LOAD Relation into Workspaces   1 &   2
LOAD Data Definition
Begin
  ASK for the tuple Key value
  SEARCH for desired tuple
  SHOW Old tuple
  ENTER new tuple
  REPLACE in workspace   2 old tuple for new one
  PRINT Workspace  2
  IF desired update THEN
  Begin
    TRANSFER Workspace   2 content to Relation File
  End
  ELSE no update made
End
```

Figure 13. Steps to update a tuple from a relation.

4. ERROR CHECKING MECHANISMS

The error checking mechanisms are present at different levels in the interaction with the system. The levels are the Data Base, the Query, the Relation, and the Interaction level. In case of some types of errors, the system responds to the user with an appropiate message. The interaction level error checking mechanisms appear in all levels and are mainly concerned with the inputs to commands and keywords to the system.

4.1. Data Base Level

The mechanisms provided at this level are put in effect when the user wants to add a new relation to the Data Base. The system will check for redundancy on the relation names. If the new relation name already exists in the data base, then the creation of the relation will be denied.

4.2. Relation Level

The error mechanisms at this level are associated with the common operations in the Data Base (tuple insertion, update,

and deletion). When the user wants to insert a new tuple, one of the first error checking mechanisms is the present number of tuples in the relation. If the tuple that is supposed to be inserted goes over the maximum number of tuples allowed, the insertion will be denied.

The other error checking mechanism appears when the actual insertion of attribute values for the new tuple is being performed. If the user inputs an attribute value that does not match the corresponding attribute domain, then the insertion is denied. The other important error checking mechanism is done to preserve the Relation integrity. This is the checking for redundant keys. If the user inputs a tuple with a key that already exists in the relation, the insertion is denied.

4.3. Query Level

The error checking mechanisms at this level correspond to the compatibility among attribute domains and values when the queries are constructed. The other error checking is using the information of the attribute value ranges. This is done when the value input for certain attribute it is not within the range that the attribute has in the data definition.

The attribute compatibility is done when the query uses two attributes, i.e. join or selection. If the attributes are not compatible in domain and length, the query is denied. The other

use of compatibility is made when a query is constructed where an attribute and a value of this attribute is placed in the query, i.e. restriction. If the value does not meet the domain and length compatibility with the attribute, the query is denied.

4.4. Interaction Level

These type of error checking mechanisms are present at the different levels of the system. There is an error checking when the user inputs the wrong attribute name and when he/she inputs a wrong relation name. When this happens, the system will give the user a chance to try the names again.

The other general error checking is done when certain answers are expected from the user, for example commands letters or Yes/No answers. The last error mechanism to mention is done at the query level; when relational operators (=,<=,<>,etc) are expected, then the system will give several chances to enter a correct relational operator.

5. SYSTEM LIMITATIONS

Limitations on the systems are caused by hardware and design objectives considerations. The limitations caused by the

hardware are those related to the size of workspaces in main memory. This implies that small relations can be brought to memory, in this case with no more than 10 tuples. The other important factor is the size of the program itself, about 1,700 lines; so there is not much space left in memory for the workspaces.

The design objectives were the development of an educational type of system that could help students studying data base management systems. The programming language used for the implementation of the Mini DBMS1 was UCSD PASCAL. The implementataion is based on the programming language facilities (data structures, built-in functions,etc) to simulate a relational data base activity. The important point was implementation of most of the relational operations. The amount of relations or tuples in them was not that important because of the type of environment for which the system was designed. The other consideration was the response time of the system to the query part or exercise of the relational operations. The system could be implemented using the diskette space for working areas, but this would lead to slower response time as a result of too much input/output activity.

The particular limitations of the system are:

* Relations cannot be more than 10 tuples each.

* Total number of relations allowed in the Data Base is 5.

* The length of each tuple cannot be more than 50 characters.

* Total number of attributes per relation is 5.

* The data types available for domain definition are strings and positive integers.

* Because of the size of the CRT (80 Characters) when a two-relations operation is performed, like Join or Cross Product, and the sum of the tuple sizes is more than 80, then the result in the last column of the CRT will be confusing. This will happen if the user is running the system in the TERAK machine; however, this will not happen if a bigger CRT or a hardcopy terminal is available.

6. CONCLUSION

This software package has been used by students in the CS482 (Spring 1980) course. They were the ones that really tested the system and made suggestions for further development, discovering some omissions that the program had. The students were asked to evaluate the system and give their ideas about it as an educational tool in the area of Relational Data Bases. The feedback obtained from the students was very interesting for the refinement of the program. Many of them found the

system helpful to understand the data base concept and exercise some of the operations in a data base, even though the relations were small.

This project has given me a very good exposure to the development of a considerable sized piece of software. This has been the result of almost a year of work, from the design of the system to the implementation and testing.

Expansions of the system can be made if a bigger minicomputer is available. This will result in bigger workspaces so that larger relations can be handled. On the other hand, if there is interest in expanding the system for real applications, then the use of bigger disks may be the way to go. For this kind of approach the basic procedures and functions developed in this program can be used. Of course, one has to consider the modification of the data structures, but the procedures for the realization of the relational operations and other utility functions can be used without modifying too much.

The other further development would be the use of this system with a CAI (Computer Aided Instruction) program in the Data Base area. The DBMS1 system can be used as a support media for the examples of relational operations and common operations in data base systems.

BIBLIOGRAPHY

* Astrahan M.M.,Blasgen M.W., et al., "SYSTEM R: Relational Approach to Database Management", ACM Transactions on Database Systems, Vol 1, #2 June 1976.

* Banerjee J. & Hsiao D.K., "Performance Study of a Database Machine in Supporting Relational Databases", Department of Computer Science, Ohio State University, 1978.

* Bernstein P.A., "Synthesizing Third Normal Form Relations from Functional Dependencies", ACM Transactions on Database Systems , Vol 1, #4, December 1976.

* Burkhard M.,"MINISEQUEL: Relational Data Management System", from "Database:Improving Usability and Responsiveness", Editor B. Shneiderman, Academic Press, 1978.

* Codd E.F., " A Relational Model of Data for Large Shared Data Banks", Communications of ACM 13, June 1970.

* Chamberlin D.D.,Astrahan M.M., et al, "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control", IBM Journal of Resesearch and Development, November 1976.

* Gagle M.,Koehler G. & Whinston A., " Data Base Systems and Micro- Computers: an overview", Krannert Graduate School of Management, Purdue University, August 1979.

* Institute for Information Systems, "UCSD Pascal Release I.4 Manual", University of California at San Diego, January 1978.

* Kroenke D.,"Database Processing", SRA Inc.,1977.

* Martin James,"Computer Data Base Organization", Prentice Hall 1975.

* Schmidt J.W.," Some High Level Language Constructs for Data of Type Relation", ACM Trans D.B.,Vol 2, #3, September 1977.

* Tsichritzis D. & Lochovsky F.," Data Base Management Systems" Academic Press 1977.

* Wirth N. & Jensen K.,"PASCAL: User Manual & Report", Springer-Verlag, 1974.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

CAPITULO D

HARDWARE

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

UN METODO CONSTRUCTIVO PARA LA DISTRIBUCION AUTOMATICA DE COMPONENTES EN CIRCUITOS IMPRESOS

Armando De Giusti
Francisco Diaz
Centro de Técnicas Analógico Digitales
de la Facultad de Ingeniería de la UNLP
La Plata, Argentina

RESUMEN:

Se presenta un algoritmo constructivo para la distribución por computadora de componentes electrónicos en una plaqueta de circuito impreso, así como el macrolenguaje utilizado en la descripción del problema.

Los aspectos más importantes del trabajo son:

- La técnica de descripción y análisis sintáctico de los datos se basa en el concepto de "línea equipotencial" y permite una fácil utilización por los no-especialistas.

- Se crea y mantiene una biblioteca de características físicas de los componentes electrónicos (dimensiones, número de terminales, orientación).

- El algoritmo de distribución de componentes se basa en un encadenamiento de criterios, todos ellos lineales con el número de componentes en tiempo de ejecución.

- Se admiten las siguientes restricciones tecnológicas: dimensiones físicas reales de los componentes, "peso" programable de las líneas, ancho mínimo de las líneas, concepto de "bus", elementos fijos y móviles, orientación preferencial de cada circuito integrado.

- Todo el sistema puede ser interactivo.

Trabajo realizado bajo la dirección del Ing. Antonio A. Quijano

INTRODUCCION:

El diseño automático de circuitos impresos y máscaras de circuitos híbridos o integrados ofrece una sucesión de problemas a tener en cuenta.

En primer término tenemos la descripción sintáctica del circuito electrónico (Ref. 1,2 y 3): se trata de desarrollar un macrolenguaje orientado utilizable por un técnico o ingeniero para describir el circuito. Deben además corregirse errores sintácticos e indicarse los parámetros que se asumen por defecto.

El segundo problema consiste en resolver la partición del sistema en subsistemas (plaquetas) y distribuir los componentes dentro de cada subsistema (Ref. 4,5,6 y 7). Se trata de problemas de asignación optimizando alguna función tal como la longitud total de conexiones en cada plaqueta ó el número de interconexiones entre subsistemas.

Por último tenemos el problema del trazado ("Layout") de las conexiones. Aquí existe un grafo que describe el circuito uniendo los terminales de componentes (vértices) con líneas equipotenciales (aristas) y es necesario hallar la representación de dicho grafo en 1,2 o más planos optimizando alguna función tal como el largo total de conexiones pesado. (Ref. 8,9,10 11 y 12).

En general estos 3 problemas abarcan disciplinas muy diferentes: computación y diseño de lenguajes, investigación operativa, teoría de grafos y computación gráfica. Además deben tenerse en cuenta las restricciones tecnológicas impuestas por el ingeniero que concibe el circuito: dimensiones físicas de componentes y plaquetas, peso relativo de la longitud de las distintas líneas de señal, espaciado entre conexiones, componentes fijos y móviles, orientación preferencial de componentes, etc..

En este trabajo presentaremos un método de descripción del circuito y un algoritmo de distribución de componentes, teniendo en cuenta las restricciones mencionadas y apuntando a satisfacer 2 objetivos principales:

1) Dar una técnica de descripción del circuito que pueda ser interactiva y resulta sencilla para el usuario.

2) Desarrollar un método constructivo (heurístico) para hallar una distribución cuasi-óptima de los componentes en el circuito electrónico que sea eficiente desde el punto de vista del tiempo de cómputo.

DESCRIPCION DEL CIRCUITO:

a) Generalidades:

En la Figura 1 se puede apreciar un diagrama en bloque del proceso de descripción y análisis.

La entrada está constituída por el archivo de componentes con los datos físicos de los mismos, la biblioteca de circuitos que pueden incorporarse como subsistemas en el problema y los datos específicos del circuito a resolver: dimensiones de la plaqueta, módulos de biblioteca y líneas equipotenciales del mismo.

La salida producida por el analizador son los mensajes de error, el listado descriptivo del circuito, el archivo de componentes actualizado y la información estructurada para ser utilizada por el programa de distribución de componentes.

Los mensajes son de 3 niveles: ALARMAS que indican que se ha omitido algún dato no imprescindible y se utilizará un valor por defecto, ERROR TIPO 1 que permite completar el análisis pero no iniciar el paso de distribución de componentes y ERROR TIPO 2 que obliga a suspender el análisis hasta ser corregido.

En la Figura 2 se puede apreciar un diagrama general del programa:

La subrutina LISTDG preprocesa la información de entrada, eliminando blancos y armando las cadenas de datos para el analizador propiamente dicho.

La subrutina ANALIZO tiene 4 bloques principales según el tipo de dato: generales, de componentes, de los buses y de cada línea equipotencial, pudiendo utilizar y actualizar la información de 2 archivos de componentes y debiendo producir los listados y la estructuración de los datos para el programa de distribución.

b) Datos a especificar en el macrolenguaje:

Los datos generales representan información no imprescindible que puede omitirse o no: nombre del circuito, dimensiones de la plaqueta, separación mínima entre líneas, opción de grabar el resultado en el archivo de circuitos.

Los datos de componentes se resumen en la Figura 3 para cada componente: tipo es una individualización genérica del grupo a que pertenece, dimx y dimy dan su tamaño relativo (Figura 4), orientación preferencial indica si alguna de las posiciones de la Figura 5 es prioritaria o ya establecida para el componente, fijo indica si ya está decidida la ubicación del componente (por ejemplo un conector) y en tal caso se especifica la misma en posición x y posición y

Los archivos de componentes (Figura 6) permiten almacenar las características correspondientes a un tipo o grupo de componentes (ARCH 1) y los nombres individuales dentro de cada grupo (ARCH 2). De este modo la especificación de un componente puede ser directa dando sólo su nombre y obteniendo la información de los archivos 1 y 2 o bien indirecta dando nombre y tipo con lo cual el sistema incorpora el nombre al archivo 2 y obtiene la información de ese tipo de componente del archivo 1. Obviamente también se pueden dar todos los datos de la Figura 3 (especificación completa) y se crean registros en los archivos 1 y 2.

Los datos de los buses comprenden una lista con los nombres y el número de líneas de cada "bus", lo cual será utilizado en la distribución y conexionado automático.

Para cada línea equipotencial se ve lo especificado en la Figura 7: K representa un número de orden del componente y los T_j son los terminales del mismo conectados a la línea. El peso da un valor relativo a la longitud de las conexiones de esa línea.

Los datos de control de macrolenguaje son los separadores parciales dentro de cada tipo de datos (: / .), entre conjuntos de datos (+) y un indicador final (*).

CRITERIOS PARA LA DISTRIBUCION DE COMPONENTES:

1) Notación:

A_j = Conjunto de los elementos colocados hasta el paso j del algoritmo.

B_j = Conjunto de los elementos que resta colocar hasta el paso j del algoritmo.

$C(x,y)$ = Conectividad (número de líneas comunes) entre los elementos x e y.

$S(x)$ = Conjunto de las líneas conectadas al elemento x.

N_k = Número de elementos de la línea K.

R_k = Costo relativo (peso) de la línea K.

L = Variable que da un peso relativo a los conjuntos de señales según su número de elementos:

$$C(x,y) = \sum R_k \ (1 + L/N_k) \ , \ K \in S(x) \cap S(y)$$

$d\left[(i,j);(i_y,j_y)\right]$ = distancia (Manhattan) del lugar (i,j) al lugar (i_y,j_y) de la plaqueta.

OR = Variable que da la orientación relativa del componente (Figura 5).

$S'(x) \equiv S_{OR,i}(x)$ = Subconjunto de señales de x que depende de la orientación OR

$S''(x) = S(x) - S'(x)$

2) Criterios:

El criterio 1 se utiliza para seleccionar el próximo elemento a incorporar a la plaqueta buscando entre los módulos no colocados aquél que tenga máxima conectividad con los ya ubicados. Se elige $\hat{x}$ tal que:

$$\sum_{y \in A_j} C(x,y) = MAX \sum_{y \in A_j} C(x,y) / x \in B_j$$

El criterio 2 nos permite decidir en qué ubicación relativa de la plaqueta conviene colocar $\hat{x}$. Para ello se analiza la conectividad componente a componente y se elige aquél $\hat{y}$ tal que:

$$C(\hat{x},\hat{y}) = MAX \left\{ C(\hat{x},y) / y \in A_j \right\}$$

El criterio 3 selecciona la posición física óptima de $\hat{x}$ al lado de $\hat{y}$ y si conviene ubicarlo horizontal o vertical (en caso de que la orientación sea totalmente libre).
Se elige el lugar $(\hat{i},\hat{j})$ tal que:

$$F3(\hat{i},\hat{j}) = MIN \left\{ F3(i,j) / (i,j) \text{ adyacente al } \hat{y} \text{ y libre} \right\}$$

donde:

$$F3(i,j) = \sum Rk\left(MIN \left\{ d\left[(i,j);(i_y,j_y) \right] , y \in K \cap A_j \right) \right\} , K \in S(\hat{x})$$

F3 evalúa el costo de conexionado del elemento $\hat{x}$ al equipotencial más cercano de sus líneas comunes con todos los elementos de Aj. Al decir que (i,j) es adyacente a $\hat{y}$ y está libre, decimos que se está considerando sucesivamente las posiciones físicas posibles para $\hat{x}$ teniendo en cuenta su tamaño y su ubicación vertical u horizontal.

El criterio 4 permite decidir cual orientación es más conveniente para el elemento $\hat{x}$. Para ello, partiendo de la posición (i,j) elegida con el criterio 3 se "despliega" el componente en 2 partes con centros a_1 y a_3 ó a_2 y a_4 según la posición a adoptar sea vertical u horizontal y se evalúa una función F4 que tiene en cuenta la orientación elegida a través del conjunto de señales S'(x) o S'' (x):

$$F4(\hat{OR}) = \underset{OR}{MIN} \; F4(OR)$$

$$F4(R) = \underset{i=1,2}{MIN} \; F'3\,(a_i, a_i+2, OR)$$

$$F'3(a_i,\ a_{i+2},\ OR) = \sum_{K \in S'(x)} Rk \Big\} \underset{y \in Aj}{MIN} \left[d(a,i);(iy,jy)\right] + \sum_{K \in S''(x)} Rk \Big\} \underset{y \in Aj}{MIN} \left[d(ai+2);(iy,jy)\right]$$

CASOS DE EMPATE:

La linealidad en el tiempo de cómputo de los criterios enunciados en el punto anterior se ve sólo comprometida por los eventuales "EMPATES" en la elección de $\hat{x}$, $\hat{y}$, $(\hat{i}, \hat{j})$ ó $\hat{OR}$. Explicaremos qué se hace en cada caso:

- Empate en la elección de $\hat{x}$: Se aplica 1 vez el criterio 2, eligiendo el $\hat{x}$ que maximice las conexiones directas con algún y de Aj:
  $\hat{x}$/ $C(\hat{x},y)$ sea máximo con $y \in Aj$

- Empate en la elección de $\hat{y}$: Se aplica 1 vez el criterio 3 a los posibles y a los que sería adyacente $\hat{x}$ y se determina simultáneamente $\hat{y}$ e $(\hat{i},\hat{j})$.

- Empate en la elección de $(\hat{i},\hat{j})$: Se aplica el criterio 4 una vez para los posibles (i,j) y se decide simultáneamente $(\hat{i},\hat{j})$ y $\hat{OR}$.

- Empate en la elección de $\hat{OR}$: Se desarrolla la ubicación del elemento $\hat{X}_{i+1}$ con los criterios 1,2 y 3 eligiendo $\hat{OR}$ de $\hat{x}_i$ tal que minimice el costo de incorporar $\hat{x}_{i+1}$. Un nuevo empate se define arbitrariamente, adoptando una dada prioridad de orientaciones.

ANALISIS DE TIEMPOS:

En los ejemplos desarrollados hasta el momento el algoritmo de distribución de componentes se ha mostrado prácticamente lineal con el número de módulos y muy rápido (Figura 8). Todos los tiempo son normalizados sobre una IBM 360/50.

A modo de ejemplo se ve en la Figura 9 el circuito de una plaqueta de 8K RAM estática, su descripción sintáctica (Figura 10) y el dibujo de la distribución de componentes resultante del algoritmo (Figura 11).

CONCLUSIONES:

Los resultados prácticos logrados hasta el momento con plaquetas de media complejidad son satisfactorios en tiempo de cómputo y utilidad de la distribución de componentes así obtenida.

La línea de trabajo actual es combinar la resolución de

los problemas de descripción y distribución de componentes con el "layout" automático e interactivo del circuito correspondiente.

BIBLIOGRAFIA:

1) Hopgood, F. - "Compiling Techniques" - American Elsevier Publishing Co - 1969

2) De Giusti A.- "Macrolenguaje y analizador sintáctico para la descripción de circuitos electrónicos" - Informe Técnico Ce. TAD - 1980.

3) Strachey, C - " A general purpose macrogenerator" - Computer Journal - Vol. 8 N°3 - 1965.

4) Brener M. "Design Automation of digital systems" - Prentice Hall - 1972.

5) Luccio y Sami - "On the decomposition of networks in minimally interconected subnetworks" - IEEE Transactions on circuit theory - Mayo 1969.

6) De Giusti y Giordana - "Montaje óptimo de plaquetas de circuito impreso" - Revista Telegráfica Electrónica - Agosto 1976.

7) Scanlon - "Automated placement of multiterminal components" - Honeywell Information Systems - 1973.

8) Lier y Otten - "Cad of masks wiring" - Report 74-E-44 Eindhoven University of Technology Netherlands.

9) Brener - "The application of integer linear programming in de sing automation" - Proc SHARE Design automation workshop- 1966

10) Hanan y Kurtzberg - A review of the placement and quadratic assignment problem" - IBM Report 1970.

11) Loberman y Weinberg - "Formal procedures for connecting terminals with a minimun total wire length" - J.ACM - Oct. 1957.

12) Bentley y Friedman - "Fast algorithms for constructing minimal spanning trees in coordinate spaces" - IEEE Transactions on computers. Feb 1978.

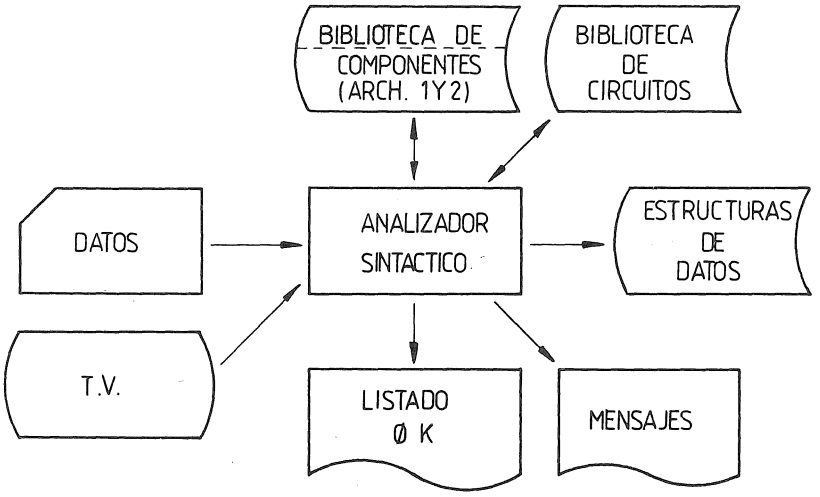


FIGURA 1

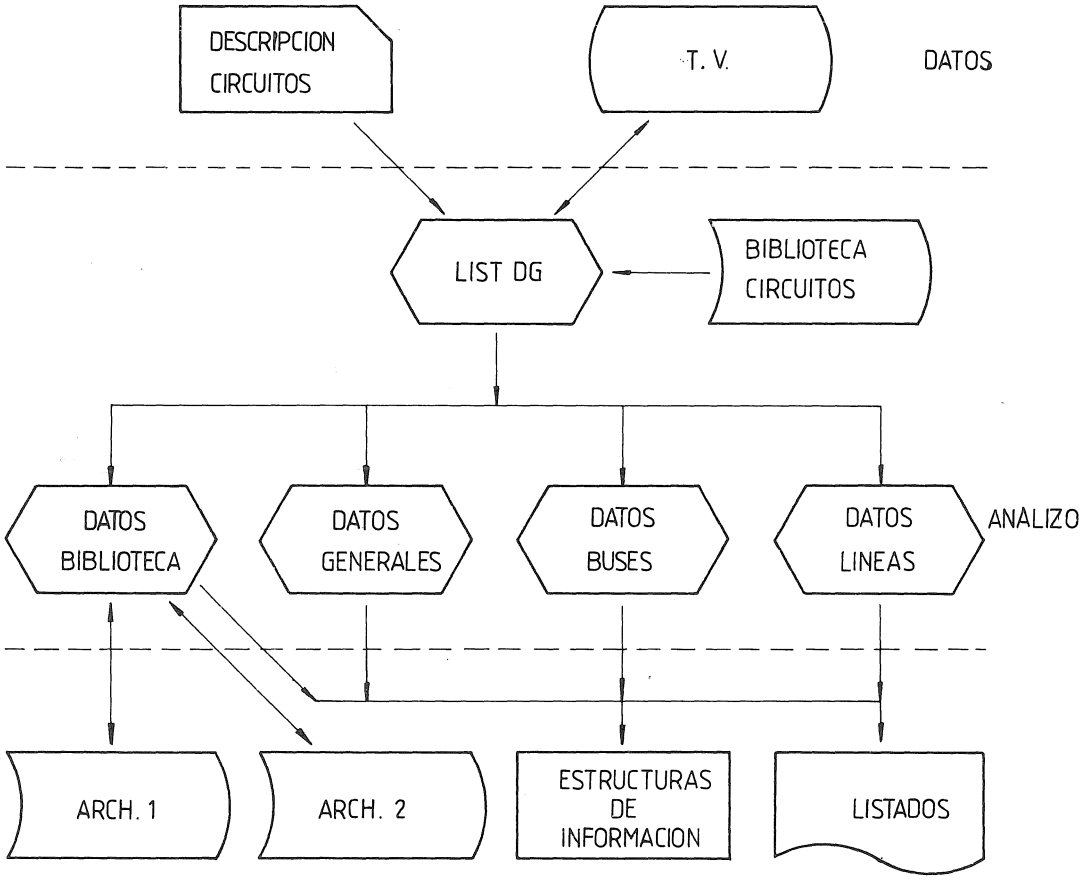


FIGURA 2

NOMBRE	TIPO	N° TERMINALES	DIMENSION X	DIMENSION Y	ORIENTACION PREFERENCIAL	FIJO	POSICION X	POSICION Y

FIGURA 3

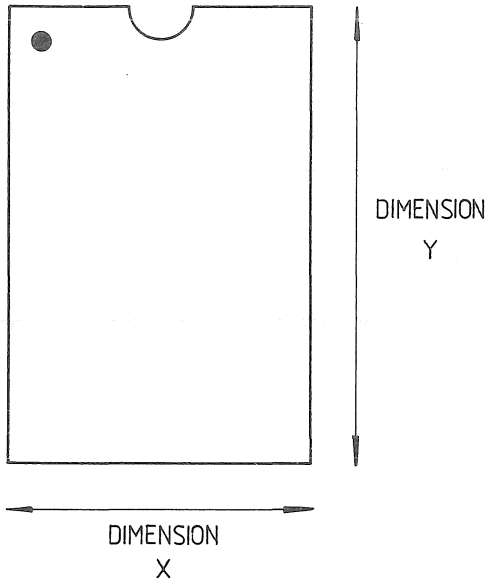


FIGURA 4

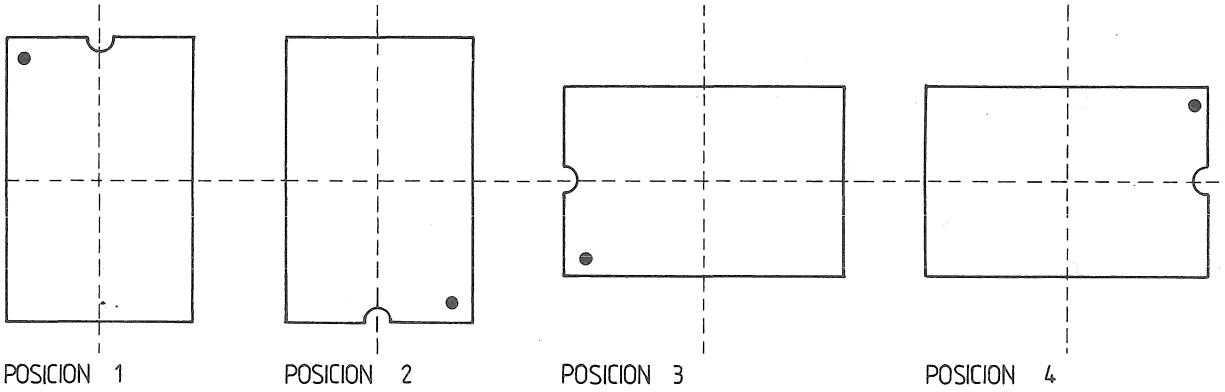


FIGURA 5

TIPO	NUMERO TERMINAL	DIMENSION X	DIMENSION Y	FIJO	POSICION X	POSICION Y

TIPO	NOMBRE 1	NOMBRE 2	.....

FIGURA 6

NOMBRE LINEA	PESO	K	Tj	K'	T'j	.....

por cada componente unido a la linea

FIGURA 7

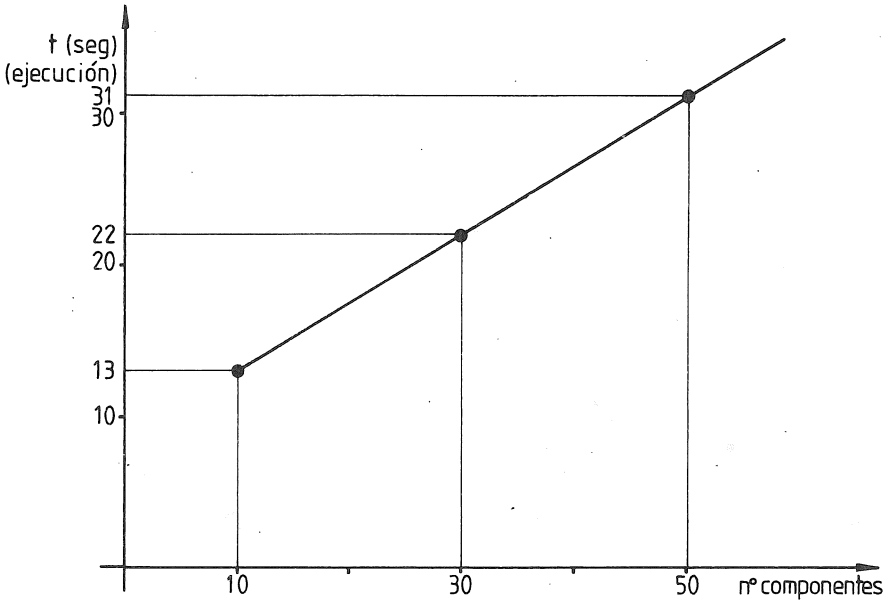


FIGURA 8

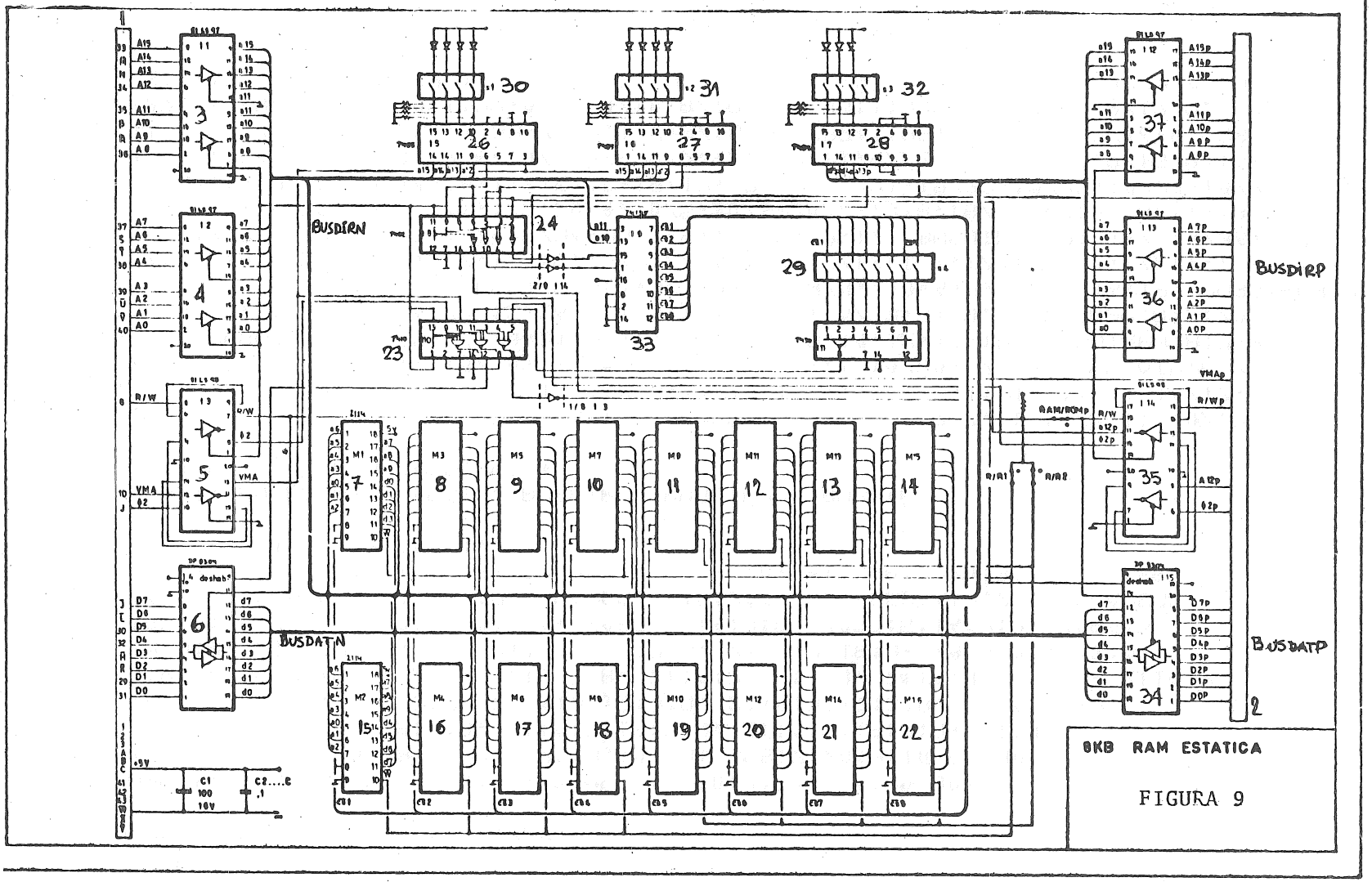


FIGURA 9

```
NOMBRE=BKRAM,DIMEX=100,DIMEY=60 +

CONEC1:1:46:2:40:1:1:10:30, CONEC2:1:40:2:30:1:1:90:30,
81LS97:2:20:3:10, 81LS97:2:20:3:10, 81LS98:2:20:3:10,DP8304:2
:20:3:10, 2114:3:18:3:8, 2114:3:18:3:8, 2114:3:18:3:8, 2114:3
:18:3:8, 2114:3:18:3:8, 2114:3:18:3:8, 2114:3:18:3:8, 2114:3:18
:3:8, 2114:3:18:8, 2114:3:18:3:8, 2114:3:18:3:8, 2114:3:18:3:8, 2114:3
:18:3:8, 2114:3:18:3:8, 2114:3:18:3:8, 2114:3:18:8, 7410:4:14:2:6,
7402:4:14:2:6, 7430:4:14:2:6, 7485:5:16:3:8, 7485:5:16:3:8, 7485:5
:16:3:8, LLAVES4:6:16:2:8, LLAVES1:6:8:2:4, LLAVES2:6:8:2:4, LLAVES3
:6:8:2:4, 74155:5:16:3:8, DP8304:2:20:3:10, 81LS98:2:20:3:10, 81LS97
:2:3:10, 81LS97:2:20:3:10 +

BUSDAT, BUSDATN, BUSDATP, BUSDIRH, BUSDIRHN, BUSDIRHP,
BUSDIRL, BUSDIRLN,BUSDIRLP +

BUSDAT:8:1,23,24,25,26,29-32/6,1-8. BUSDATN:8:6,12-19/7,11-14/8,11-14/19,
11-14/10,11-14/11,11-14/12,11-14/13,11-14/14,11-14/15,11-14/16,11-14/17,
11-14/18,11-14/19,11-14/20,11-14/21,11-14/22,11-14/34,12-19. BUSDATP:8,2,1-8/34
,1-8. BUSDIRH:8:1,33-36,20,21,22,19/3,2,4,6,8,14,12,16,18. BUSDIRHN:8:3,3,5,7,11
16/18,15,16/19,15,16/20,15,16/21,15,16/22,15,16/33,3,13. BUSDIRHP:8:2:24-30
/37,2,4,6,8,13,15,17. BUSDIRL:8:1,37-40,15,18/4,2,4,6,8,12,14,16,18. BUSDIRLN:8:
1-7,17/13,1-7,17/14,1-7,17/15,1-7,17/16,1-7,17/17,1-7,17/18,1-7,17/19,1-7,17/20
,1-7,17/21,1-7,17/22,1-7,17/36,35,7,9,11,15,13,17. BUSDIRLP:8:2,17-24/36,2,4,6
,8,12,14,16,18. VCC:1:1,1/3,20/4,20/5,20/6,20/7,18/8,18/9,18/10,18/11,18/12,18/
13,18/14,18/15,18/16,18/17,18/18,18/19,18/20,18/21,18/22,18/23,14/24,14/25,14/26
44/3,1,10/4,10/5,10,19/6,10/7,9/8,9/9,9/10,9/11,9/12,9/13,9/14,9/15,9/16,9/17,9/
L11:1:32,7/28,12. CS1:1:33,7/29,1/7,8/15,8. CS2:1:33,6/29,2/8,8/16,8. CS3:1:33,
5/29,3/9,8/17,8. CS4:1:33,4/29,4/10,8/18,8. CS5:1:33,9/29,5/11,8/19,8. CS6:1:33,
NCS1:1:29,1/33,1.NCS2:1:29,2/33,2. NCS3:1:29,3/33,3. NCS4:1:29,4/33,4. NCS5:1:29
```


Figura 10

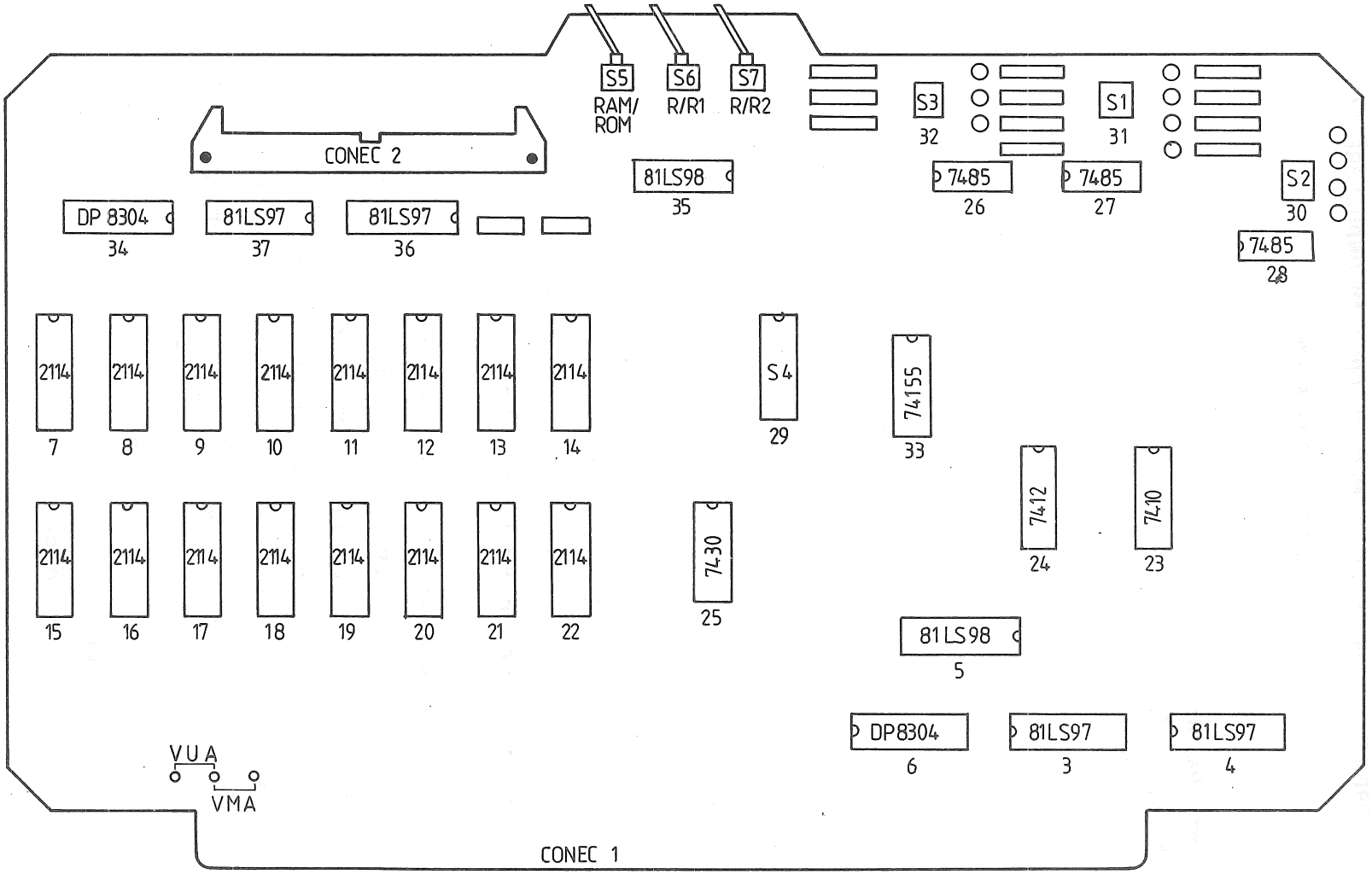

S5
RAM/
ROM
S6
R/R1
S7
R/R2
S3
32
S1
31
S2
30
CONEC 2
81LS98
35
7485
26
7485
27
7485
28
DP 8304
34
81LS97
37
81LS97
36
2114
7
2114
8
2114
9
2114
10
2114
11
2114
12
2114
13
2114
14
S4
29
74155
33
7412
24
7410
23
2114
15
2114
16
2114
17
2114
18
2114
19
2114
20
2114
21
2114
22
7430
25
81LS98
5
DP8304
6
81LS97
3
81LS97
4
VUA
VMA
CONEC 1

FIGURA 11

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

PROJETO DE CONSTRUÇÃO DE MEMORIA COM SEMICONDUCTORES PARA COMPUTADOR B-6700

Newton Braga Rosa
Fernando Rosa Do Nascimento
Cilmo Oliveira
Universidade Federal Do Rio Grande Do Sul
Centro de Processamento de Dados
Rua Ramiro Barcelos SN
90 000 — Porto Alegre — RS, Brasil

APRESENTAÇÃO

O objetivo mais imediato deste projeto é a construção de memória principal, em semicondutor , para o Computador de grande porte, BURROUGHS B-6700.

O projeto está sendo desenvolvido por duas universidades brasileiras com tradição em projetos de "hardware" de computadores : Universidade Federal do Rio Grande do Sul(UFRGS) e Fundação Universidade de Brasília (UnB).

Estas duas instituições, independentemente, a partir de 1975, já construíram diversos dispositivos para ampliação da capacidade computacional de seus computadores B-6700. Tais dispositivos estão funcionando regularmente e sua construção permitiu acumular uma quantidade considerável de conhecimentos técnicos a respeito daqueles computadores. Recentemente a UnB projetou e construiu uma memória de 48kB para seu processador de comunicação de dados (DCP). A experiência dos projetos anteriores, aliada à capacidade de engenharia das duas universidades, levou ao atual projeto, o qual é de complexidade e repercussão bastante maiores.

Os primeiros estudos sobre a viabilidade técnica do empreendimento iniciaram em setembro de 1979, a partir da constatação, por um lado dos problemas enfrentados pelas instituições para a expansão de seus equipamentos B-6700 e por outro das diretrizes governamentais para o setor de informática. Estas diretrizes são estabelecidas pela Secretaria Especial de Informática vinculada ao Conselho de Segurança Nacional. O ato normativo nº 1/80 (anexo 1) traz consigo o espírito do principal objetivo da SEI, que é a capacitação nacional em "Hardware" e "Software" na área de informática.

O projeto fci detalhado vários meses após a realização dos primeiros contatos que envolveram, além das duas universidade , a companhia BURROUGHS. Esta última ficou encarregada de colocar a disposição das duas universidades toda documentação de interesse ao projeto. A partir da análise desta documentação foi possível definir o projeto com precisão, estimando custos, prazos e , principalmente , cotejando a sua complexidade com a capacidade e recursos das duas Universidades. Concluiu-se, em resumo, que o projeto é perfeitamente factível.

JUSTIFICATIVA

Vários aspectos justificam este projeto; todos eles, entretanto , decorrem da política governamental, cujo principal objetivo é dotar o País de uma maior capacitação tecnológica neste setor, considerado de segurança nacional.

Em termos mais específicos o projeto se justifica a partir dos elementos abaixo:

a) Existem 22 computadores BURROUGHS B-6700 no País em 16 instalações diferentes.

Os primeiros computadores BURROUGHS B-6700 foram instalados no País por volta de 1971. A UFRGS foi a primeira Universidade a adquiri-lo e, a partir desta experiência, várias outras instituições passaram a usá-lo. Esta máquina tornou-se dominante no cenário nacional de computadores de grande porte em ambiente universitário, durante toda a década de 70. Neste período, uma série de outras instituições passaram a usar este equipamento que foi responsável, inclusive, por uma mudança na imagem da Companhia BURROUGHS, anteriormente voltada para equipamentos de escritório e computadores de pequeno e médio porte.

b) Praticamente todos os computadores B-6700 instalados no País estão limitados pela sua capacidade de memória principal.

Originalmente o computador B-6700 foi projetado para usar memória de núcleo de ferrite. O núcleo de ferrite é um pequeno anel de material ferro-magnético, no interior do qual passam no mínimo três fios. Os fios identificam individualmente o núcleo e fazem as operações de leitura e escrita. A construção de tais módulos é quase artesanal e muito sujeita a falhas, sendo, por estas razões, seu custo muito alto. Atualmente o emprego de núcleos de ferrite diminui em favor de tecnologias moder-

nas, seguras e baratas. Devido ao alto preço da memória de ferrite , praticamente todos os computadores B-6700 estão limitados por memória principal. Em outras palavras, estas máquinas teriam o seu rendimento sensivelmente melhorado caso recebessem mais memória. Com relação a UFRGS e a UnB, ambas universidades encontram-se próximas dos limites máximos de utilização dos seus computadores. Este aumento de memória permitirá resolver parcialmente o problema permitindo, ainda, a expansão das operações em telerpocessamento.

c) Os computadores B-6700 instalados no País estão virtualmente impedidos de crescerem.

Devido a significativas vantagens das memórias de semicondutores, as memórias de núcleo de ferrite deixaram de ser fabricadas. Eventualmente, se algum usuário conseguir localizar no mercado internacional um módulo de memória disponível, sua aquisição implicará em evasão de divisas. Por outro lado o usuário, além de estar pagando em preço 10 vezes superior ao de um módulo de igual capacidade, em tecnologia de semicondutores, estará sujeito a um longo processo para efetivar a importação.

d) É necessário aumentar a vida útil e capacidade dos computadores instalados no País , com o mínimo possível de importações .

Computadores B-6700 representam um considerável patrimônio. São máquinas de excelente desempenho, de arquitetura muito avançada e em condições de prestar bons serviços por muitos anos como ocorrem com outras máquinas do mesmo fabricante em operação a mais de 15 anos (B-500, B-3500 e B-3700). Por outro lado, os equipamentos periféricos que estão sendo fabricados no Brasil (discos e impressoras) são compatíveis com o B-6700. Logo, justificam-se plenamente os esforços para construção de memórias que possam ampliar a capacidade computacional e a vida útil destes equipamentos.

e) Os conhecimentos adquiridos serão úteis aos esforços governamentais de capacitação tecnológica da indústria nacional de informática.

A partir de 1978, o Governo incentivou a fabricação de minicomputadores por empresas legitimamente nacionais. Estas empresas compraram tecnologias no exterior e assumiram o compromisso de, gradativamente, assimilarem aquela tecnologia, ao mesmo tempo que adaptariam ao novo produto, os avanços de eletrônica digital.

Desta forma, os conhecimentos adquiridos com este projeto poderão reverter em benefício da indústria nacional pois, ao seu final , vários técnicos de duas universidades terão acumulado conhecimentos importantes para o projeto e construção de memória de semicondutor para os minicomputadores nacionais.

f) Os conhecimentos adquiridos poderão servir a outros computadores importados já instalados no País.

A Companhia BURROUGHS possui dezenas de computadores no Brasil (modelos B-3500 e B-3700), que utilizam memória de ferrite semelhante a do B-6700. Assim os conhecimentos adquiridos poderão ser úteis em projetos semelhantes voltados para outros computadores deste ou de algum outro fabricante.

OBJETIVOS

OBJETIVO GERAL

O objetivo principal deste projeto é colocar a capacidade de projeto e de desenvolvimento de duas universidades, com experiência sobre o computador B-6700, a serviço dos objetivos governamentais estabelecidos para o setor de informática.

OBJETIVOS ESPECÍFICOS

a) Adquirir conhecimentos a respeito de desenvolvimento de memória de semicondutor (LSI) para computadores de grande porte.

Posteriormente os conhecimentos adquiridos poderão ser repassados aos fabricantes de minicomputadores nacionais.

b) Ampliação da memória dos computadores B-6700 da Universidade Federal do Rio Grande do Sul e da Universidade de Brasília; está prevista a construção de 4 módulos de 128 Kwords (4x800 kBytes) para cada Universidade. Estes módulos funcionarão como protótipos para testes de desempenho, confiabilidade e durabilidade.

c) Fabricação desta memória, em escala semi-industrial, para estender as vantagens decorrentes do emprego da memória de semicondutor a outros usuários de sistemas B-6700.

CARACTERÍSTICAS DAS MEMÓRIAS DE SEMICONDUTORES (LSI)

Atualmente, a melhor solução para armazenamento primário onde seja necessário tempos de acesso menor que um microsegundo, está no uso de circuitos monolíticos integrados em larga escala. Esta tecnologia permite a construção das memórias chamadas de RAM (Randon Acess Memory) do tipo dinâmica.

Os progressos têm sido consideráveis refletindo-se no aumento da escala de integração e na diminuição do preço por bit (já existem projetos de pastilhas com até 512 Kbits).

Embora esta tecnologia já estivesse sendo empregada em microprocessadores, somente no início de 1979, os fabricantes lançaram no mercado computadores de grande porte, usando memória de semicondutores (LSI): IBM 4341; BURROUGHS B-6800; DIGITAL Vax-11/780; Honeywell Bull 60/66 DPS, etc.

Todos estes computadores se caracterizam pela sua larga capacidade de memória principal que lhes confere capacidade de processamento excepcionais com relação às máquinas de "geração" anterior. O custo da memória se tornou tão pequeno, com relação ao resto do equipamento, que a maioria já vem com uma quantidade de memória alcançada somente em configurações excepcionais grandes da "geração" anterior. Assim, é comum encontrar computadores com 2 ou mesmo 4 Megabytes de memória principal. Por outro lado, constata-se que, via de regra, as memórias de semicondutor das novas máquinas são incompatíveis com os modelos mais antigos. Em resumo, os aspectos que justificam o uso da memória de semicondutor dinâmica no lugar da memória de núcleos de ferrite são os seguintes:

a) Custo substancialmente menor

b) Alta integração; isto significa que o espaço ocupado e por conseguinte o gabinetes são consideravelmente menores.

c) Menor consumo de energia. Consumo de energia proprocional ao uso e ao tamanho da memória; menor que o de núcleo de ferrite para capacidades iguais.

d) Menor dispersão de calor.

e) Maior facilidade de manutenção; na eventualidade de algum problema é necessário tão somente a troca de uma pastilha afixada sobre uma placa de circuito impresso. Nas memórias de ferrite, qualquer problema por menor que seja, obriga o envio de todo o módulo de volta para a fábrica.

f) Existe no mercado mundial uma grande quantidade de fornecedores de memória LSI, ao contrário do que ocorre com as memórias de ferrite. Constata-se também, uma perfeita compatibilidade entre os produtos de fabricantes diferentes o que significa, em termos práticos, menores custos para o consumidor e garantia de pronta substituição ao longo do tempo.

g) O custo das memórias LSI dinâmica de 16Kbx1 está decrescendo rapidamente.

DESCRIÇÃO DO PROJETO

Como o projeto de memória de semicondutor com circuitos integrados com larga escala de integração (LSI) é dirigido ao computador de grande porte BURROUGHS B-6700,o módulo de memória deve obedecer aos sinais padrões e ser funcionalmente idêntico aos tipos de memórias usadas no B-6700. Os módulos de memória são ligados ao MC III (Memory Control Model III).

No projeto será usada uma memória dinâmica com 16K x 1 bit e tempo de acesso de 200 nanosegundos. Sua escolha foi decorrência da alta densidade de bits por integrado e do baixo custo por bit, em relação às memórias estáticas.

MEMORY CONTROL III

E um controlador de memória do B-6700 que interliga processadores e módulos de memória; tem por função reconhecer comandos e endereços dos processadores, gerar sinais de controle aos módulos e resolver conflitos no caso de acessos simultâneos por mais de um processador.

O MC III é basicamente uma matriz " crosspoint" de seis entradas e quatro saídas. Cada entrada pode ser ligada a um processador, multiplexor ou testador de memória; cada saída pode ser ligada a um módulos de 16Kw, 32Kw, 64Kw ou ainda 128Kw. Portanto um único MC III pode controlar até 512 Kw (4 x 128 ou 4 x 800 Bytes), ou seja, metade da capacidade máxima de endereçamento do B-6700).

Um diagrama em bloco do MC III está representado na figura 1.

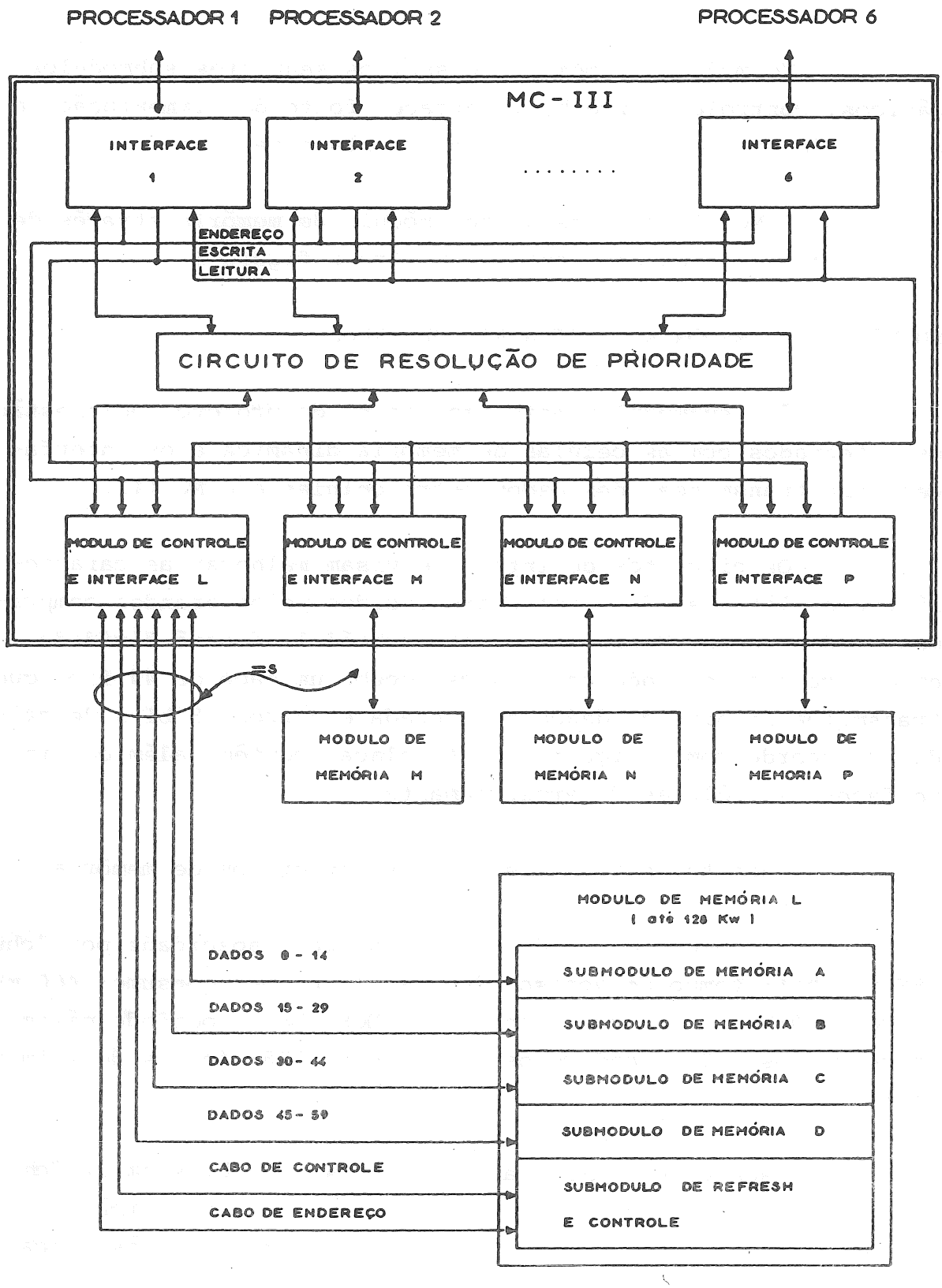


Fig.1 - DIAGRAMA EM BLOCOS DO MC III COM OS NOVOS MODULOS.

MÓDULO DE MEMÓRIA

Um módulo de memória possui os seguintes submódulos básicos: controle, refresh, interface, fonte de alimentação e células de armazenamento (memória propriamente dita).

O MC III se liga a cada módulo de memória através de um conjunto de 6 cabos, conforme mostra a figura 1.

SUBMÓDULO DE MEMÓRIA E INTERFACE DE DADOS

É o submódulos mais importante do projeto. Nele estão os integrados com as células de memória dinâmica e os interfaces para transmissão dos dados entre células e o MC III.

Os circuitos de interface visam melhorar as características elétricas dos sinais distorcidos pelos grandes comprimentos dos cabos. Fisicamente este submódulo ficará dividido em quatro placas, onde cada placa recebe um cabo de 44 fios que transmitem 15 bits de dados de entrada e outros 15 bits de saída, de acordo com a figura 1-b. As placas contêm, além dos interfaces, as células de armazenamento.

Características físicas dos integrados de memória:

As memórias são do tipo dinâmica; capacidade por "chip" 16K x 1 bit; tempo de acesso: 200 ns; 16 pinos; consumo: 460 mW ativo e 20mW em repouso; tecnologia: MOS (LSI); período máximo entre refresh: 2ms; endereços de refresh: 128; fontes de alimentação: 3.

Cada placa armazenará até 128Kw x 15 bits em 120 "chips". Assim, uma palavra de 60 bits é formada pela justaposição de 4 conjuntos de 15 bits, obtidos pelo acesso simultâneo às placas (128kW = 800kB).

SUBMÓDULO DE REFRESH E CONTROLE

Recebe 2 cabos de MC III: um de endereço (até 17 bits) e um de controle. A lógica de controle identifica se a operação é de leitura, leitura/escrita, leitura/modifica/escrita ou refresh. As três primeiras são solicitadas pelos processadores e a última é gerada internamente.

Devido ao tipo de memória empregada (dinâmica), os integrados devem ser refrescados periodicamente (2ms em 2ms) em todos os 128 endereços de linha. Estas operações devem ser intercaladas com as operações normais vindas dos processadores. A lógica de controle será baseada numa máquina de estados, operando em alta velocidade, pois os tempos envolvidos são da ordem de dezenas de nanosegundos.

A lógica de controle se encarrega ainda da verificação de paridade dos bits de endereço e controle vindos do MC III.

SUBMÓDULO DA FONTE DE ALIMENTAÇÃO

Fornece 3 tensões de alimentação ao módulo de memória. Contém cicuitos que prevêem o acionamento seqüenciado das várias tensões envolvidas durante as operações de "power-on" e "power-off" do computador. O seqüenciamento das fontes é necessário para evitar danos às memórias dinâmicas de tecnologia MOS.

Não é possível e nem aconselhável usar as fontes de alimentação existentes no B-6700, pois além da incompatibilidade de nível haveria sobrecarga e possíveis interferências.

SUBMÓDULO DE REFRIGERAÇÃO

Como as necessidades de refrigeração nos submódulos de memória MOS são pequenas (≃ 60W/placa) optou-se por usar o sistema de refrigeração existente, durante a fase de desenvolvimento de protótipo.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

SINTESIS DE CONTADORES CON SOLO ELEMENTOS DE MEMORIA

J. Aguiló

E. Valderrama

R. Escardó

Departamento de Informática
Facultad de Ciencias
Universidad Autónoma de Barcelona
Bellaterra (Barcelona), España

1.- INTRODUCCION

La síntesis de contadores síncronos, entendidos estos como máquinas secuenciales autónomas de comportamiento periódico, ha sido estudiada desde distintos puntos de vista debido a las numerosas aplicaciones de este tipo de autómatas en los sistemas digitales en general. Se ha prestado especial interés a la síntesis de contadores mediante interconexión de elementos de memoria exclusivamente; no sólo por el atractivo que representa construir contadores a partir de elementos idénticos (razón que tal vez ha quedado obsoleta debido al rápido avance tecnológico y al abaratamiento de los chips), sinó también por la simplificación del conexionado que representa, y el aumento de velocidad (y fiabilidad) que se produce al desaparecer el conexionado inherente a las puertas. El fin primordial de estos trabajos ha sido, en consecuencia, el de conocer "a priori" cuales son las secuencias de estados que pueden implementarse con un número mínimo de biestables y sin puertas (2),(3) y (4).

Para ello, salvo alguna excepción, el método de estudio ha consistido en una búsqueda exhaustiva previa de éstas secuencias para posteriormente inferir, en lo posible, resultados de tipo general.

En este artículo se lleva a cabo un desarrollo matemático que nos permitirá calcular, para cada valor de n, las longitudes de las distintas secuencias que se pueden implementar con n biestables y sin lógica combinacional. Dicho desarrollo se ha realizado en principio para biestables tipo D, aunque nos permitirá inferir algunos resultados para otros tipos de flip flops, como veremos más adelante.

2.- DEFINICIONES

Los conceptos que se definen a continuación son harto conocidos; sin embargo, hemos considerado necesario este primer apartado de definiciones por cuanto sienta las bases de todo el desarrollo posterior.

2.1.- MAQUINAS SECUENCIALES AUTONOMAS

Una máquina secuencial autónoma (msa.) (clasicamente un cuadriplete $<Q,S,\delta,\lambda>$, $Q=\{\text{estados}\}$, $S=\{\text{salidas}\}$, δ: función estado siguiente y λ: función de salida), es una aplicación $F=(f_1,f_2,\ldots,f_n)$ de B^n en B^n $(2^{n+1}<\#(Q)\leq 2^n)$, de la forma:

$$F : B^n \longrightarrow B^n$$

$$x \longrightarrow F(x)=(f_1(x),f_2(x),\ldots,f_n(x))$$

donde $f_i : B^n \longrightarrow B \qquad \forall i\in\{1,2,\ldots,n\}$

Nótese que el conjunto de los estados es un subconjunto de B^n $(Q\subset B^n)$, y que no consideramos las salidas del autómata. Esto es perfectamente válido puesto que estamos interesados en las longitudes de ciclo, y no en los ciclos en sí.

Supuesta la síntesis con biestables tipo D, las aplicaciones f_i son funciones de conmutación que representan las funciones de entrada a cada uno de los biestables que componen la máquina. En consecuencia, las funciones f_i deben ser literales (variables complementadas o no), o las constantes 0 o 1 para que la máquina no posea lógica combinacional.

2.2.- M.S.A. COSTE NULO

En otras palabras, una msa. tendrá coste cero (se define el coste en función de la lógica combinacional presente), cuando las funciones f_i sean de la forma:

- $f_i = \varepsilon_j^{\alpha}$ $\{i,j\} c \{1,\ldots,n\}$; $\alpha \epsilon \{0,1\}$
  ε: función proyección
- $f_i = 0^{\alpha}$ $(0^0=1,\ 0^1=0)$

2.3.- REPRESENTACION MATRICIAL DE MSA. COSTE CERO

Dichas funciones $F: B^n \longrightarrow B^n$ pueden generalizarse a funciones lineales $F: R^n \longrightarrow R^n$ (ver (5)), de forma que toda msa. coste cero con biestables D puede representarse por una matriz M nxn de 0,1 y -1 con un elemento no nulo por fila a lo sumo, y tal que:

- $m_{ij}=1 \Rightarrow f_i=\varepsilon_j$
- $m_{ij}=-1 \Rightarrow f_i=\overline{\varepsilon}_j$
- $m_{ij}=0 \quad \forall j \Rightarrow f_i=0$

Nota: Para hacer esto es necesario pasar a una representación de $B=\{1,-1\}$ en vez de la $\{0,1\}$ habitual; de forma que ahora al hablar, por ejemplo, del estado 5 (101), lo representaremos por 1-11. Asimismo, el caso $f_i=1$ no está previsto puesto que se puede demostrar que todo autómata conteniendo una de las funciones $f_i=1$ es isomorfo a otro con $f_i=0$.

3.- CONTADORES

Hemos caracterizado, hasta ahora, las msa. coste cero. Todas ellas, por tener un número finito de estados, presentan un comportamiento al menos parcialmente periódico; esto es, en su grafo de comportamiento aparece al menos un ciclo cerrado (en el peor de los casos un ciclo de longitud 1). Nuestra tarea siguiente consistirá en identificar dichos ciclos de forma que, dado un contador de una determinada longitud, podamos predecir el mínimo número de biestables que serán necesarios para su síntesis sin puertas. Para ello, dividiremos el estudio en dos partes:

3.1.- MSA. COMPLETAS

Las msa. completas son aquellas en las que la función F es una biyección de B^n en B^n. Se demuestra facilmente (ver apéndice 1), que la condición necesaria y suficiente para que F sea biyectiva es que las funciones f_i sean de la forma:

$$f_1=\varepsilon_{i_1}^{\alpha_1}, f_2=\varepsilon_{i_2}^{\alpha_2}, \ldots, f_n=\varepsilon_{i_n}^{\alpha_n}$$

con, $\{i_1,i_2,\ldots,i_n\}\subset\{1,2,\ldots,n\}$; $i_j\neq i_k \quad \forall j\neq k$

Este tipo particular de máquinas es especialmente fácil de tratar por cuanto, al ser F una biyección, el grafo de comportamiento está exclusivamente formado por ciclos cerrados, y en consecuencia puede estudiarse como un elemento del grupo simétrico S_{2n}. En este sentido, podemos hablar de descomposición en ciclos de una msa. completa.

De acuerdo con la definición, la matriz asociada a una msa. completa será una matriz nxn, de 0,1 y -1, con un elemento no nulo por fila y por columna.

El estudio de las longitudes de ciclos en este caso veremos que es especialmente simple. Para ello es necesario ver

previamente el siguiente teorema, cuya demostración se da en el apéndice 2:

Teorema. Dos msa. completas cuyas matrices asociadas posean el mismo número de menores de dimensión i ($\forall i \in \{1,2,\ldots,n\}$) con determinante +1 y el mismo número con determinante -1 poseen la misma descomposición en ciclos. (Sólo tenemos en cuenta los menores centrados en la diagonal que no contienen menores de dimensión más pequeña).

En consecuencia, a partir de este punto, tomaremos como iguales aquellas máquinas cuyas matrices cumplan la condición anterior, aunque rigurosamente hablando sean isomorfas.

Consideraremos dos casos:

3.1.1.- MSA. COMPLETAS COMPUESTAS

Toda máquina cuya matriz asociada posea más de un menor con determinante no nulo es una máquina compuesta (de hacho, es una composición paralelo de msa. más simples).

En efecto, la existencia de k menores no nulos y que no contengan ningún menor más pequeño (puesto que evidentemente dos menores de orden k_1 y k_2 pueden formar un menor de dimensión k_1+k_2), de dimensiones $i_1,i_2,\ldots,i_k$ ($\sum_{j=1}^{k} i_j = n$), significa que la máquina F se compone de k submáquinas $F_1,F_2,..,F_k$ disjuntas, cada una de ellas conteniendo $i_1,\ldots,i_k$ biestables interconectados entre sí. A título de ejemplo, la matriz:

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

posee un menor de dimensión 1 ($m_{22}=1$), y dos menores de dimensión 2 (m_{13},m_{31}; y m_{45},m_{54}). La máquina representada por esta

matriz, en consecuencia, estará compuesta por tres submáquinas de 1,2 y 2 biestables respectivamente como puede observarse en la figura 1.

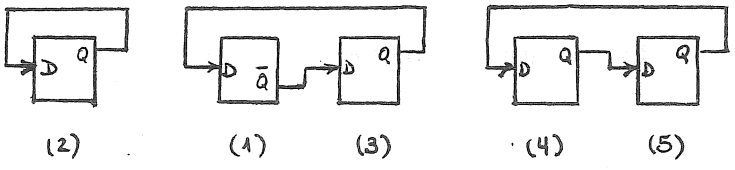


Fig. 1

Suponiendo que pudiésemos conocer las longitudes máximas de cada una de las k submáquinas, resulta claro que la longitud de ciclos máxima de la máquina compuesta sería el m.c.m. de las longitudes de cada submáquina; ya que cada estado de F es una k-epla $(s_1,s_2,\ldots,s_k)$, donde $s_1,\ldots,s_k$ son los estados de $F_1,F_2,\ldots,F_k$. El siguiente paso, por lo tanto, consistirá en hallar las longitudes de ciclos máximas correspondientes a las máquinas que llamaremos simples.

3.1.2.- MSA. COMPLETAS SIMPLES

Son aquellas cuya matriz asociada posee un sólo menor de dimensión n. De acuerdo con el teorema anterior, sólo existirán dos descomposiciones en ciclos diferentes; la correspondiente a un menor con determinante +1, o con determinante -1. Puesto que la longitud de ciclos máxima coincide con el orden de la matriz, concluimos que toda msa. completa simple de dimensiones rxr (r biestables) genera, o bien una longitud de ciclo máxima de r, o bien de 2r.

El siguiente teorema nos asegura que no hemos de preocuparnos por las longitudes no máximas:

Teorema. a) En una cierta descomposición en ciclos de una msa. completa, todas las longitudes de ciclosque aparecen son divisores de la longitud máxima.

b) Para un cierto valor de n, las longitudes máximas que aparecen en las distintas descomposiciones en ciclos son tales que si aparece la longitud l_i, también aparecen todas las longitudes l_j divisoras de l_i.

Demostración. a) La primera parte es trivial.

b) La segunda parte puede demostrarse por inducción. En efecto; se cumple para n=1, cuyas dos únicas longitudes máximas son 1 y 2, y para n=2 cuyas longitudes son 1,2 y 4. Supongamos que se cumple para $n \leq x$. En n=x+1 aparecerán:

- Todas las matrices de n=x con un ±1 adicional en la diagonal; esto es, todas las longitudes de la forma:

  $mcm(1_i^x, 1) = 1_i^x$, que cumple la condición por hipótesis.

  $$mcm(1_i^x, 2) = \begin{cases} 1_i^x & \text{si } 1_i^x \text{ es múltiplo de 2} \\ 21_i^x & \text{si } 1_i^x \neq \dot{2} \end{cases}$$

  En este último caso deberían aparecer además las longitudes 21_j^x con 1_j^x divisor de 1_i^x; pero todas ellas aparecerán ya que si $1_i^x \neq \dot{2} \Rightarrow 1_j^x \neq \dot{2} \Rightarrow mcm(1_j^x, 2) = 21_j^x$.

- Las matrices simples de dimensión x+1 que producirán las longitudes (x+1) y 2(x+1). Puesto que por hipótesis se cumple para n x, aparecerán todas las longitudes de 1 a x, y en consecuencia, (y puesto que el divisor mayor de 2(x+1) es (x+1)), todos los divisores de (x+1) y 2(x+1).

3.1.4.- ALGORITMO DE BUSQUEDA DE LONGITUDES

En virtud de las consideraciones anteriores, y teniendo en cuenta que sólo necesitamos buscar las longitudes máximas, estamos ya en disposición de presentar un algoritmo que determine cuales son las secuencias que pueden generarse con biestables D y sin lógica combinacional.

Paso 1. Descomponer el número n en sumandos de todas las formas posibles. Existen N descomposiciones diferentes, con

$$N = \sum_{k}(-1)^{k-1}.p\left(n - \frac{3k^2 \pm k}{2}\right); \quad p(0)=1;\ p(n)=N;\ 1<\frac{3k^2 \pm k}{2} \leq n$$

Esta descomposición nos dará la estructura de menores de la máquina.

Paso 2. Para cada descomposición de n ($n = \sum_{i=1}^{k} s_i$), cada uno de los sumandos nos produce, sea la longitud s_i, sea la $2s_i$.

Paso 3. Las longitudes máximas se hallarán buscando en mcm de todas las posibles k-eplas $(s_1^*, s_2^*, \ldots, s_k^*)$, donde por s_i^* indicamos s_i o $2s_i$ indistintamente. El proceso debe realizarse para las N descomposiciones de n.

EJEMPLO : Búsqueda de las longitudes para n=4.

1) Existen 5 posibles descomposiciones de 4, que son:

$4 = 4$
$1+3$
$2+2$
$1+1+2$
$1+1+1+1$

2) Los menores de dimensión 1 producirán las longitudes 1 y 2; los de 2 las 2 y 4; los de 3 las 3 y 6; y el de 4 las 4 y 8.

3) En consecuencia, las longitudes para n=4 serán:

Descomp. de n	Longitudes
4	4 y 8
1+3	mcm(1,3)=3; mcm(1,6)=6; mcm(2,3)=6; mcm(2.6)=6.
2+2	2 y 4
1+1+2	2 y 4
1+1+1+1	1 y 2

En resumen, el n=4 genera las longitudes 1,2,3,4,6 y 8.

En la tabla siguiente se muestran los resultados obtenidos para $1 \leqslant n \leqslant 13$.

n	Longitudes
1	1 2
2	1 2 4
3	1 2 3 4 6
4	1 2 3 4 6 8
5	1 2 3 4 5 6 8 10 12
6	1 2 3 4 5 6 8 10 12
7	1 2 3 4 5 6 7 8 10 12 14 20 24
8	1 2 3 4 5 6 7 8 10 12 14 15 16 20 24 30
9	1 2 3 4 5 6 7 8 9 10 12 14 15 16 18 20 24 30 40
10	1 2 3 4 5 6 7 8 9 10 12 14 15 16 18 20 21 24 30 40 60
11	1 2 3 4 5 6 7 8 9 10 11 12 14 15 16 18 20 21 22 24 28 30 40 56 60
12	1 2 3 4 5 6 7 8 9 10 11 12 14 15 16 18 20 21 22 24 28 30 35 40 42 56 60 84 120
13	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 24 26 28 30 35 36 40 42 44 48 56 60 72 80 84 120

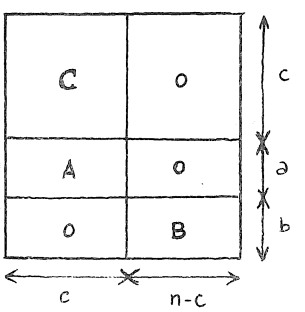
Tabla I. N°de biestables D necesarios para implementar con coste 0 contadores de una cierta longitud.

3.2.- MSA. NO COMPLETAS

Las msa. no completas son aquellas en las que F no es una biyección. Para estas máquinas se cumple el siguiente teorema:

Teorema.Para un cierto valor de n, las longitudes de los ciclos generadas por las msa. no completas son un subconjunto del de las completas.

Demostración.Sea M la matriz asociada a una msa. no completa. M puede escribirse (reordenando filas y columnas) como:



$n = c+a+b$

donde C contiene todos los posibles menores no nulos; A es una matriz con un ±1 por fila a lo sumo; y B tiene asimismo un ±1 por fila como máximo, y al menos una columna de ceros. La longitud máxima nos vendrá dada por un r tal que $M^r(x)=x\ \forall x\epsilon B^n$. En general, M^i tendrá el siguiente aspecto:

C^i	0
$A.C^{i-1}$	0
0	B^i

B tiene al menos una columna de ceros, y por tanto, B^i poseerá al menos i columnas de ceros. Así, para i=n-c (o i n-c si B tiene más de una columna de ceros), $B^{n-c}=0$. Evidentemente ninguno de los productos M^i cumple que $M^i=M$, ya que $B^i\neq B$ (de hacho, nos estamos moviendo sobre los caminos no cíclicos del grafo). A partir de este índice i=n-c es cuando podemos buscar los ciclos. Tenemos ahora una matriz M' con B=0, y buscamos un r tal que:

C	0
A	0
0	0

=

C^r	0
AC^{r-1}	0
0	0

Si $C^r=C \Rightarrow C^{r-1}=I \Rightarrow AC^{r-1}=A \Rightarrow (M')^r=M'$. Luego la longitud máxima viene fijada única y exclusivamente por los menores C. Pero evidentemente, los menores son los mismos que aparecían en el caso de los completos, con lo cual no podrá aparecer ninguna longitud nueva.

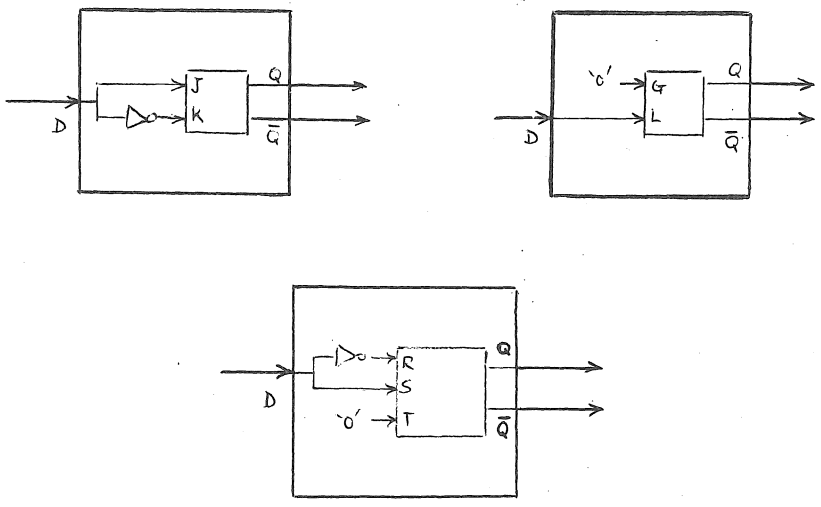
En consecuencia, podemos afirmar que la tabla 1 contiene todas las longitudes de ciclo que se pueden generar con n biestables tipo D y sin lógica combinacional.

4.- SINTESIS CON OTROS TIPOS DE FLIP FLOPS

Las siguientes propiedades permiten inferir ciertos resultados sobre la síntesis de contadores con otros tipos de biestables que no sean los D.

1. El flip flop RS es funcionalmente equivalente al D (1), y en consecuencia la tabla de longitudes 1 es perfectamente válida para el RS. Por funcionalmente equivalente entendemos la propiedad de los flip flops D y RS demostrada en (1) según la cual toda msa. sintetizable con n flip flops D y sin puertas puede ser implementada con el mismo número de flip flops RS y coste cero, y viceversa.

2. El número de flip flops JK, GL o RST necesarios para implementar una cierta msa. es menor o igual al número de flip flops D. La tabla de longitudes, por tanto, nos da una cota máxima del número de flip flops JK, GL o RST imprescindibles en la síntesis de un contador dado. Esta propiedad es evidente por cuanto el comportamiento de un flip flop D puede ser simulado por un JK, GL o RST sencillamente tomando $J=\overline{K}=D$; $G=1$ $L=D$; y $T=0$ $\overline{R}=S=D$, como puede verse en la fira siguiente:



APENDICE 1

TEOREMA

Una función $F=(f_1,f_2,\ldots,f_n) : B^n \longrightarrow B^n$, con $f_i=\varepsilon_k^{\alpha j}$ o $f_i=0^{\alpha j}$ es biyectiva $\Leftrightarrow$ $f_i=\varepsilon_j^{\alpha k}$ con $\alpha_k \neq \alpha_1 \quad \forall k \neq 1$.

DEMOSTRACION

1- F biyectiva $\Leftrightarrow$ $f_1^{\alpha 1}.f_2^{\alpha 2}\ldots f_n^{\alpha n} \neq 0 \quad \forall(\alpha_1,\alpha_2,..,\alpha_n)\epsilon B^n$

2- Si $f_i=0^{\alpha j} \Rightarrow f_1^{\alpha 1}\ldots(0^{\alpha j})^{\alpha j}\ldots f_n^{\alpha n}=f_1^{\alpha 1}\ldots 0\ldots f_n^{\alpha n}=0$

En consecuencia las f_i constantes hemos de omitirlas.

Si $f_i=\varepsilon_j^{\alpha k} \quad \forall i \Rightarrow$

$$f_1^{\alpha 1}..f_n^{\alpha n}=(\varepsilon_{j1}^{\alpha j1})^{\alpha 1}\ldots(\varepsilon_{jn}^{\alpha jn})^{\alpha n}=\varepsilon_{j1}^{\alpha j1 \equiv \alpha 1} \qquad \varepsilon_{jn}^{\alpha jn \equiv \alpha n}$$

Evidentemente esta expresión es distinta de cero para cualquier valor de $\alpha_1\ldots\alpha_n \Leftrightarrow \alpha_{ji} \neq \alpha_{jk} \quad \forall i \neq k$.

APENDICE 2

Llamemos I_n al cojunto de las funciones $F=(f_1,f_2,\ldots,f_n)$ con $f_i=\varepsilon_k^{\ j}$ biyectivas. En I_n establecemos la siguiente relación

$$F_1,F_2 \epsilon I_n \qquad F_1 \sim F_2 \Leftrightarrow \exists \sigma \epsilon I_n \mid \sigma^{-1}.F_1.\sigma = F_2$$

Esta relación:

Esta relación:

1- Es una relación de equivalencia.

2- La partición inducida en I_n es evidentemente más fina que la inducida por la isomorfía de grafos. (En tal caso es necesario unicamente que σ sea biyectiva, pero no que pertenezca a I_n).

TEOREMA

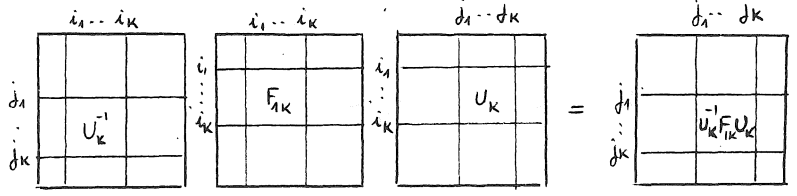
Dos msa. completas cuyas matrices asociadas posean el mismo número de menores de dimensión i ($i\epsilon\{1,2,\ldots,n\}$) con determinan

te +1 y el mismo número con determinante -1 tiene la misma descomposición en ciclos.(Sólo consideraremos los menores centrados en la diagonal que no contienen menores más pequeños).

DEMOSTRACION.

La demostración se hará por construcción viendo que si F_1 y F_2 son dos matrices con las características antes citadas, entonces $\exists U \in I_n$ tal que $U^{-1}.F_1.U=F_2$. La construcción de esta matriz U se hará en varios pasos:

1. Supongamos que exista un menor de dimensión k que en F_1 ocupa las filas y columnas $i_1,i_2,..,i_k$, y en F_2 las $j_1,j_2,...,j_k$. U deberá tener un menor no nulo en las posiciones $i_1,..,i_k,j_1,...,j_k$, como indica la figura.



Este proceso se sigue para todos los menores de F_1 y F_2.

2. El menor $U_k^{-1}.F_{1k}.U_k$ todavía no tiene porqué coincidir con el menor de dimensión k F_{2k}.(coinciden unicamente en posición). Sean F_{1k} y F_{2k} dos menores de la misma dimensión y con el mismo determinante:

a) Si $(F_{1k})_{ij}$ y $(F_{2k})_{ij}$ son no negativos $\forall i,j$, dichos menores pueden verse como matrices de incidencia de grafos isomorfos, y en consecuencia $\exists U_k \mid U_k^{-1}.F_{1k}.U_k=F_{2k}$.

b)En caso contrario, el proceso se realiza en dos partes, aprovechando la propiedad de las matrices de I_n en virtud de la cual toda $U \in I_n$ puede descomponerse en el producto $U=D.|U|$ donde D es tal que $d_{ii}=1$, $d_{ij}=0$ $\forall i \neq j$, y $|U|$ significa el valor absoluto de U . El apartado (a) nos asegura que $\exists U$ tal que $|U|^{-1}|F_{1k}||U|=|F_{2k}|$. Luego:

$$(|U^{-1}|\cdot F_{1k}|U|)_{ij} = (F_{2k})_{ij}$$

Por tener el mismo determinante, si llamamos n_1 al número de elementos -1 de $|U^{-1}|F_{1k}|U|$ y n_2 al de F_{2k}, $n_1=n_2\pm 2x$ $x\in N$.

b.1) Si $n_1=n_2$ y supongamos que $(F_{1k})_{ij}=-1$, $(F_{2k})_{ij}=1$, y $(F_{1k})_{1m}=1$, $(F_{2k})_{1m}=-1$. En general, llamaremos r' a la columna de F_{1k} o F_{2k} en la que la fila r posea un ± 1. Hacemos $d_{ii}=d_{i'i'}=\ldots=d_{11}=-1$ y el resto $d_{jj}=1$. Siempre podremos llegar al índice 11 puesto que F_{1k} y F_{2k} son menores no nulos centrados en la diagonal. Con este valor de D se cumple (tengamos en cuenta que multiplicar una matriz a derecha e izquierda por D equivale a cambiar de signo las filas y columnas i para las cuales $d_{ii}=-1$):

$$(D.F_{1k}.D)_{ij}=(F_{2k})_{ij}$$
$$(D.F_{1k}.D)_{1m}=(F_{2k})_{1m}$$

b.2) Si $n_1=n_2+2x$ $x\in N-\{0\}$, en virtud del paso anterior, los -1 de F_{1k} pueden agruparse de forma que tengamos 2x elementos tales que $(F_{1k})_{ii'}=1$ y $(F_{1k})_{i'i''}=1$. Haciendo $d_{ii}=-1$ en cada uno de los casos conseguiremos aumentar en 2x el número de -1 de la matriz F_{1k}, y pasar por tanto al caso anterior $n_1=n_2$.

BIBLIOGRAFIA

(1) Aguiló,J.:"Circuitos secuenciales sintetizados exclusivamente con flip flops." Tesis Doctoral. Universidad Autónoma de Barcelona, Setiembre 1977.

(2) Acha J.I.,Huertas J.L., Merino J.:"Generación y clasificación de circuitos contadores sin puertas con 4 biestables JK." Revista de Informática y Automática, vol 40, pp 11-22. 1979.

(3) Davio M., Bioul G.:"Interconnection structures of injective counters composed entirely of JK flip flops." Information and Control, vol 33, pp 304-332. 1977.

(4) Manning F.B.,Fenichel R.:"Synchronous counters constructed entirely of JK flip flops." IEEE Trans. on Computers, vol C-25, pp 300-306. March 1976.

(5) Valderrama E.:"Asignación de estados en máquinas sin lógica combinacional." Tesis Doctoral. Univ. Aut. de Barcelona. Junio 1979.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

MICROPROCESADOR EN LA ADQUISICION Y PROCESO DE DATOS DE TEMPERATURA AMBIENTE

J. González Hernando
J. Alberdi Primicia
J. Sanchez Izquierdo
Sección de Sistemas de Control
División de Instrumentación y Control
JUNTA DE ENERGIA NUCLEAR
Av. Complutense 22, Madrid, España

INTRODUCCION

En determinado tipo de instalaciones, como pueden ser las nucleares, resulta necesario un conocimiento continuado de la temperatura ambiente y su evolución a lo largo del año.

Por motivos:

a) de situación geográfica, relativos a la ubicación del futuro Centro de Soria, y

b) de tipo económico,

se ha buscado la solución en un sistema totalmente automático y autónomo en el que la intervención de un operador se ha reducido al mínimo.

El sistema que vamos a describir realiza de forma automática :

- La tarea de toma de datos de temperatura ambiente
- Manipulación y análisis en tiempo real de los datos primarios
- Obtención de datos derivados
- Presentación adecuada de los datos.

Expondremos en breve análisis del problema y su solución, una descripción somera del equipo físico y un esquema

.../...

de la operación, tal como viene gobernada por el programa.

1. ESPECIFICACIONES DEL SISTEMA

Las especificaciones del sistema consisten pues, fundamentalmente en:

1) Suministrar información escrita de forma automática

cada hora sobre:

- La temperatura instantánea

cada día sobre:

- La temperatura media con tomas cada diez minutos
- La temperatura máxima del día y hora de ocurrencia
- La temperatura mínima del día y hora de ocurrencia
- La oscilación entre la máxima y la mínima.

cada mes sobre:

- Temperatura media
  media máxima
  media mínima
  media oscilaciones
  máxima y hora
  mínima y hora

- Valor de la oscilación máxima y día

y los mismos cada año referidos al año.

2) Responder a un mínimo de requerimientos de usuario como son:

- ajuste de reloj
- modificación de hora
- petición de información de temperatura.

Para cumplir el objetivo propuesto el sistema posee la capacidad de almacenamiento de los datos relativos a un año completo. En los casos en los que no se necesita una individualización del dato, el almacenamiento se realiza de forma acumulativa con el consiguiente ahorro de memoria.

Para garantizar el funcionamiento general del sistema, y en particular la conservación de información en memoria

volátil se utiliza como protección la alimentación a través de acumuladores. En caso de fallo de la red el sistema sigue alimentado y funciona normalmente con autonomía suficiente.

El ciclo básico de operación se ha establecido en los 10 minutos, teniendo en cuenta el valor relativamente alto de la constante de tiempo asociado a la temperatura ambiente.

De esta forma resulta un sistema con gran elasticidad en cuestiones de tiempo. Los cálculos a realizar consisten fundamentalmente en obtención de valores medios, comparaciones de búsqueda (máximos y mínimos) y los de cambio de código o representación. No presentan problemas especiales,por lo que resulta un sistema holgado de tiempo.

2. EQUIPO FISICO

El sistema se apoya en el microprocesador MC6800 de Motorola, al que vá conectada la tarjeta de memorias y los circuitos de E/S (fig. 1).

El sensor empleado es una resistencia variable -- con la temperatura que produce una señal analógica recogida en un convertidor A/D de 16 líneas de datos BCD. Un elemento de - entrada realiza las operaciones de interfase para que el dato pase de los registros al micro.

La información se suministra a través de:

1) Una impresora de 21 caracteres por línea e impresión elec-- trostática sobre papel metalizado,

y

2) Una ventana de visualización formada por 6 elementos de 7 - segmentos.

El elemento de comunicación hombre-máquina es un teclado de 10 teclas numéricas y 10 de función.

2.1. DISPOSITIVO Y LOGICA DE IMPRESION

Como dispositivo de salida de datos, comunicación de mensajes y petición de parámetros, se ha seleccionado la impresora MEGAPRINT serie MP300. Las razones principales de esta elección han sido, reducido tamaño, sencillez de mecanismo e - impresión electrostática sobre papel metalizado.

Asociada a esta impresora se dispone de una lógica que recibe, almacena los datos y los convierte en señales de impresión. Asimismo, genera señales relativas a su estado y recoge las líneas de órdenes y control. La lógica diseñada a este fin, tiene como diagrama bloque el presentado en la figura 2, y a ella nos referimos para su explicación cualitativa.

La pieza fundamental del cirucito de control la constituye un generador de caracteres que consta de una memoria ROM (2240 bits), un contador y un decodificador de dirección. Las entradas se toman de los 6 dígitos que forman un carácter ASCII y las salidas actúan sobre los elementos de impresión formados por una matriz de electrodos de 7 x 5.

La entrada de datos y la impresión resultan totalmente asíncronas por la existencia de una memoria FIFO que independiza los procesos.

Los 6 bits de información aplicados a la entrada se almacenan en la memoria mediante una señal de transferencia (Shift in).

La impresión se produce al llenarse la línea o cuando aparece la señal de impresión (Print).

Para acomodar la asincronía de los dos procesos se generan en la lógica una serie de señales de estado. Como más adecuada a nuestro modo de operación y sencilla de manejo, hemos seleccionado la señal de motor ocupado/dispuesto (Busy motor).

También se utiliza la señal de borrado de memoria para evitar interferencias entre líneas de distinta longitud.

Para la conexión al micro (Fig. 3) se ha empleado únicamente un subconjunto de las señales de control y estado suficiente para asegurar la operación. Por eso, basta con la utilización de un único lado del elemento de E/S (PIA Pheripheral Interchange Adapter), teniéndose disponibles las señales de control y estado que gobiernan la secuencia de impresión.

Se ha utilizado la línea de entrada CA1 de la PIA para recibir el estado de motor dispuesto/ocupado como única señal que se recibe de la impresora.

La orden de impresión, como más fundamental, se envía por la línea CA2 en modo de salida y las señales de in--

troducción en memoria y borrado de la misma por las líneas de datos 6 y 7 respectivamente. Las restantes 6 líneas de datos (0 a 5) se utilizan para salida de los 6 dígitos de un carácter ASCII.

La conexión se hace a través de alimentadores de línea que invierten la señal, hecho que debe ser tenido en -- cuenta en el programa.

2.2. TECLADO-VISUALIZADOR

Este periférico de E/S, basado en el KIT II de -- Evaluación de Motorola, ha sido transformado en un dispositivo de introducción de parámetros y de petición de datos y listado.

El teclado funciona sin interrupciones y por un - método de escrutado. El dispositivo de representación visual - comparte sus líneas de selección de dígitos con las de selección de fila en la matriz del teclado (fig. 4).

Este hecho no tiene reprecusión desde un punto de vista físico, pero sí habrá quetenerlo en cuenta con las rutinas de manejo.

2.3. TERMOMETRO Y CONVERTIDOR A/D

Se utiliza el lector digital de temperatura PI4453 de Analogic. Sus características principales son:

- resolución de 0.1º C con $\pm$ 0,05% de precisión digital.
- 16 líneas de salida en paralelo de dígitos decimales codificados BCD.
- líneas de Polaridad y de Sobrepasamiento.
- señal de Fin de Conversión.
- señales de control.

La conexión al microprocesador se hace a través de una PIA tal y como indica la figura 5.

La sincronización de la toma del dato se realiza - a partir de la señal de fin de conversión que entra por una de las líneas de recepción de interrupciones (CA1).

La señal de polaridad se conecta como bit de signo al bit más significativo de las líneas de entrada de datos.

2.4. OSCILADOR

Para marcar la secuencia de medida ajustando la frecuencia de muestreo se dispone de un oscilador que produce una interrupción a través de la línea CB1 de la PIA del termómetro cada segundo.

3. OPERACION

3.1. PROGRAMA EJE

La operación del sistema viene organizada alrededor de la interrupción producida por el oscilador cada seg.

El servicio a la interrupción actualiza la hora y la fecha, y examina si se ha llegado al límite de los 10 minutos, del día, del mes o del año para proceder en consecuencia.

Cuando cesan las actividades enumeradas anteriormente, el programa se dedica a refrescar el visualizador y a examinar el estado del teclado para atender una eventual acción sobre el mismo.

Los programas de iniciación sitúan los periféricos, interfases, registros calendarios, etc, de manera que la operación puede comenzar de forma correcta.

El programa se ha escrito en ensamblador y en fortran. La filosofía seguida ha sido la utilización de alto nivel, en este caso fortran, siempre que fuera posible, sin un excesivo coste de complejidad y complicación.

La estructura general del sistema de programación se presenta en la fig. 6. En las figuras 7 y 8 se tienen los programas de iniciación y de servicio a la interrupción.

3.2. TOMA DE DATOS

El convertidor A/D asociado al termómetro, está continuamente efectuando la conversión digital a un ritmo de 24 c/s para mantener actualizado el dato en su propio visualizador.

Por eso, cuando se necesita tomar un dato, el programa espera a la próxima señal de fin de conversión para leer el registro de la PIA. A continuación se manipula el dato para

entregarlo con un formato adecuado para ser tratado por las rutinas de cálculo escritas en fortran.

El dato entra formando un doble octeto BCD con la indicación de signo en el bit más significativo.

En las figuras 9 y 10 se presenta diagrama bloque de la rutina de toma de datos y tratamiento de los mismos.

3.3. IMPRESION DE LINEA

Una vez programado el circuito de E/S (PIA), cada vez que se desea imprimir una línea se llama a la rutina IMP de impresión de línea. Su diagrama bloque se presenta en la fig. 11. Unas pocas indicaciones bastarán para dejar suficientemente claro su significado.

Comienza IMP analizando el estado de la impresora esprando que el motor esté dispuesto.

A continuación se vá dando salida a cada carácter seguido de una orden de almacenamiento.

Cuando se ha terminado con el último carácter se genera una orden de impresión y se retorna al programa que llamó a la subrutina.

La fig. 12 presenta el diagrama bloque de la rutina de listados.

3.4. PROGRAMA DE TECLADO/VISUALIZADOR

Como ya hemos mencionado antes, no se utilizan interrupciones para el control de estos periféricos.

La filosofía de funcionamiento se basa en un escrutado de la matriz del teclado y en un refresco del dispositivo de visualización en los tiempos muertos entre actividades de toma de datos y tratamiento de los mismos.

Las rutinas están basadas en las utilizadas por el monitor JBUG del KIT II de evaluación adaptadas a las necesidades de nuestra aplicación.

3.4.1. REFRESCO DEL DISPOSITIVO DE VISUALIZACION

Esta rutina recoge el dígito a representar, busca

el código correspondiente para 7 segmentos, seleciona el número de orden y dá salida al código.

Su función se presenta en el diagrama bloque de la figura 13.

3.4.2. ESCRUTADO DEL TECLADO

La rutina selecciona mediante un direccionamiento por fila y columna la tecla correspondiente y prueba la línea testigo de tecla pulsada.

Si la línea está activa, pasa control a la rutina de decodificación y distribución de funciones.

El direccionamiento del punto correspondiente se hace situando los bits del registro de salida de la PIA. Este está dividido en dos partes. Los 6 dígitos menos significativos se corresponden cada uno de ellos con una fila de matriz.

Los dos dígitos más significativos dan el código de una de las cuatro columnas de la matriz.

Las operaciones de la rutina se presentan en el diagrama bloque de la fig. 14.

3.4.3. RUTINA DE DECODIFICACION Y DISTRIBUCION

La filosofía de decodificado se basa en la comparación con una tabla de patrones que sirven para identificar la tecla pulsada.

Se hace una primera distinción entre número y letra para poder distinguir de forma inmediata valores numéricos de parámetros.

Si es letra, lo que significa petición de una función determinada, se debe efectuar la distribución del flujo de control a la rutina correspondiente.Para ello se utiliza el método de una tabla de instrucciones de salto a la que se accede mediante el contenido de un contador de teclas.

En la figura 15 se tiene un diagrama bloque de la rutina.

4. CONCLUSION

Como conclusión vamos a enunciar unos resultados que nos ha dado la experiencia en este proyecto.

- No resulta rentable trabajar con "kits" preparados para -- evaluación como base del diseño del equipo. Es más eficaz el diseño de tarjetas a medida de la aplicación.

- El compilador Fortran suministrado por Motorola, adolece de

  i) poco potente, lo cual es enteramente explicable para un micro, pero lo apuntamos

  ii) tiene un factor de expansión altísimo. En nuestra aplicación tenemos en cuanto a ocupación de memoria:

      RAM 1 K
      EPROM 1 K programas en ensamblador
            9 K programas Fortran y Librería Fortran
            (4,2 programas y 4,8 librerías).

  iii) a veces se han tenido problemas de depuración al tener - que seguir la expansión, llamadas y encadenamiento.

Por eso quizás se justifica un uso amplio del ensamblador reservando el Fortran únicamente para rutinas de -- cálculo.

- Las interrupciones añaden siempre complejidad y sofistica-- ción al programa. Por eso parece aconsejable, dada la dedicación del sistema, el utilizarlas lo imprescindible, bus-- cando métodos alternativos de escrutado o pregunta al periférico.

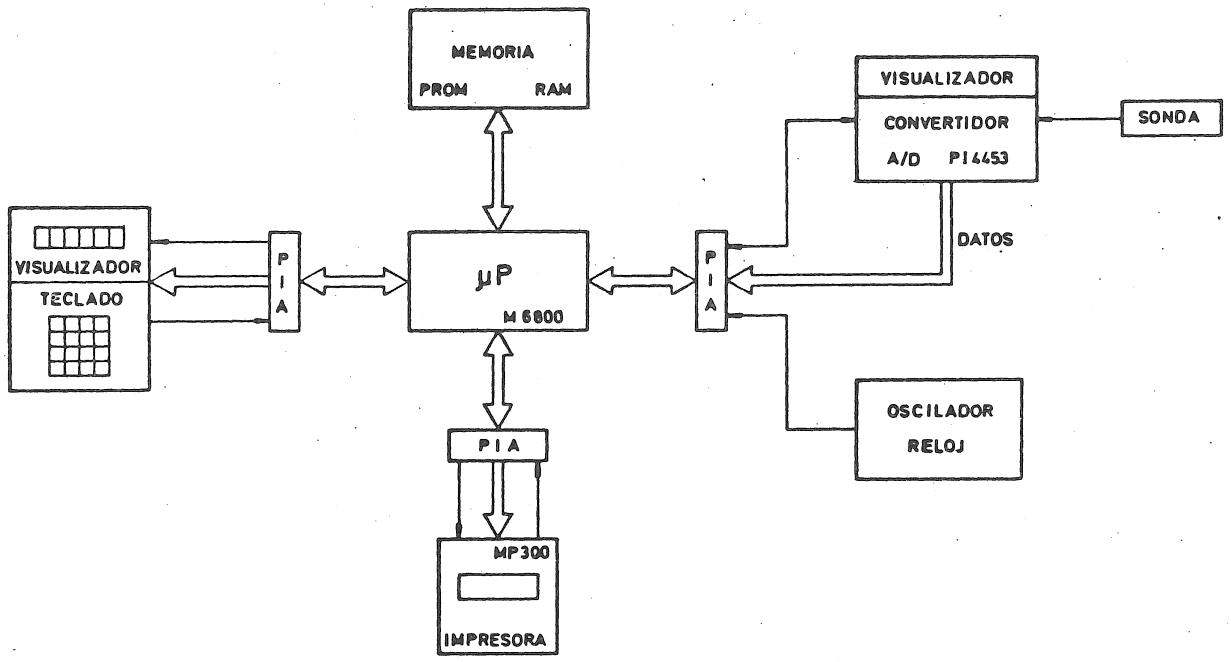


Fig._1

DIAGRAMA DEL SISTEMA

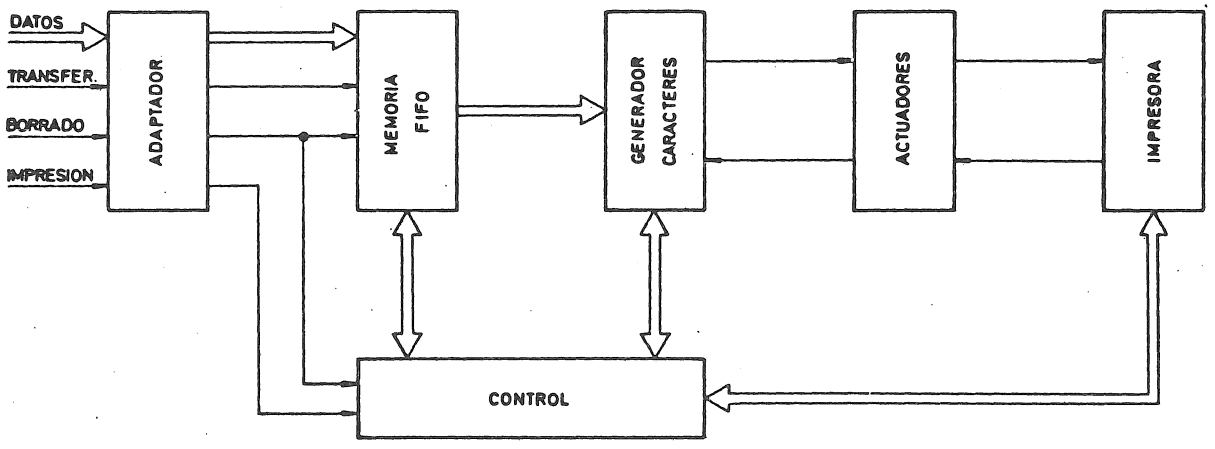


Fig._2

CONTROL DE LA IMPRESORA

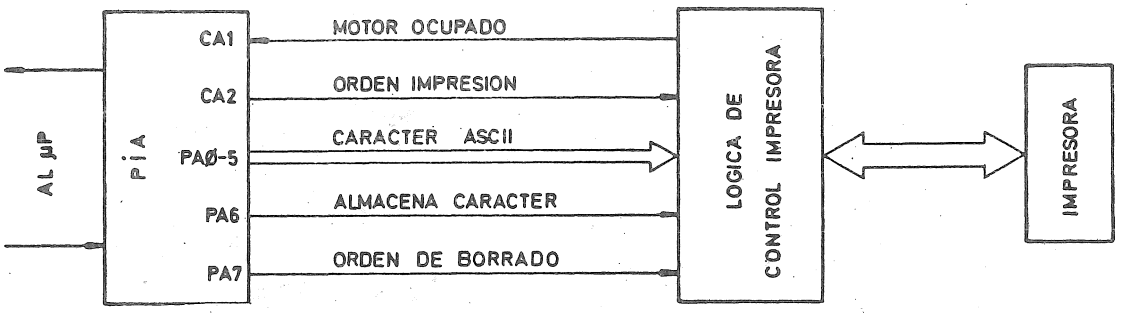


Fig._3

CONEXION DE LA LOGICA DE
CONTROL DE LA IMPRESORA

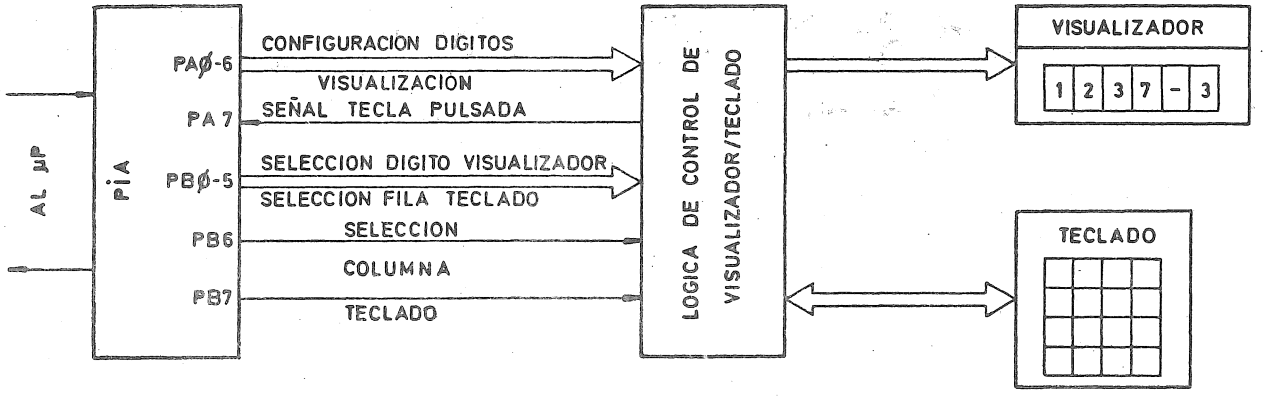


Fig._4

CONEXION DE LA LOGICA DE CONTROL
DEL VISUALIZADOR/TECLADO

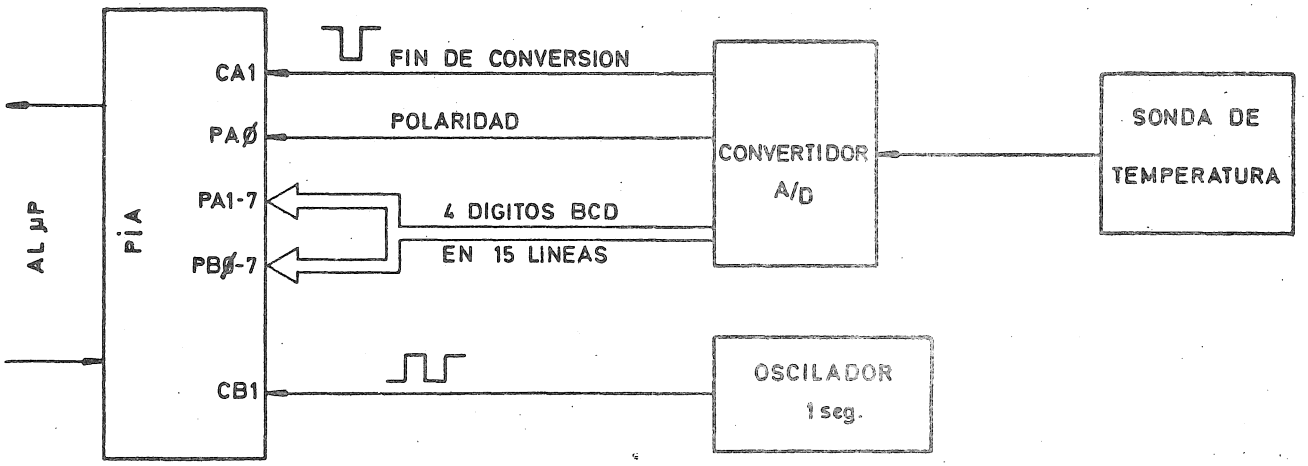


Fig._5

CONEXION DE ENTRADA DE DATOS DE TEMPERATURA

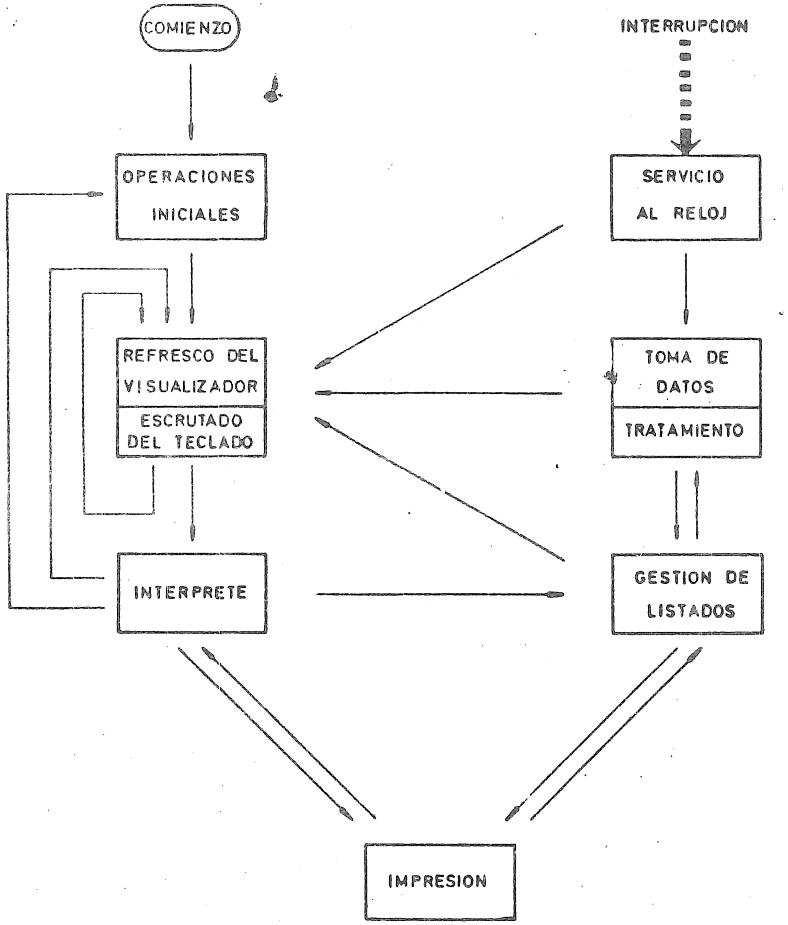


Fig._6

ESTRUCTURA DEL SISTEMA
DE PROGRAMACION

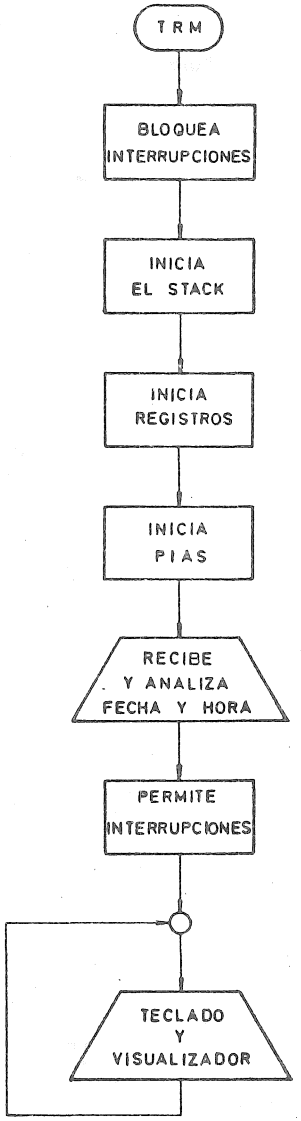


Fig._7

PROGRAMA DE PREPARACION
DE LA OPERACION DEL SISTEMA

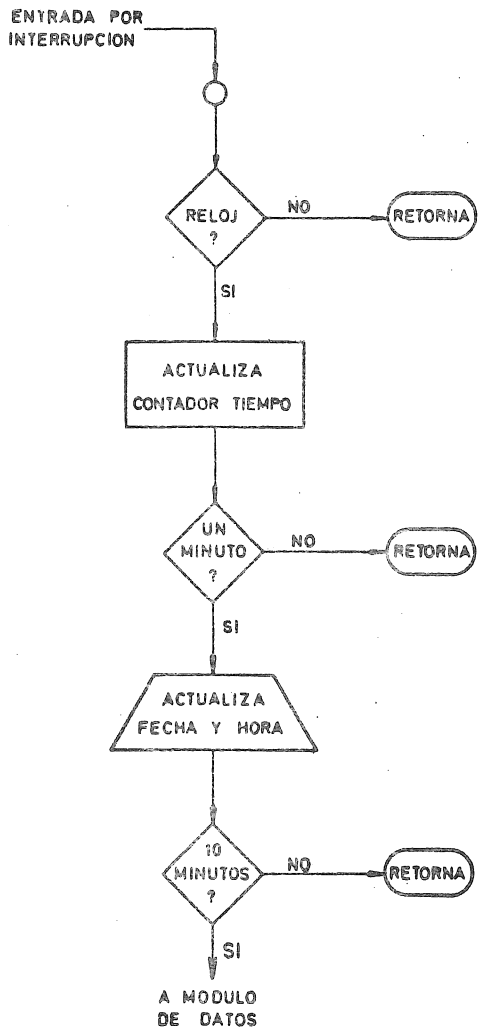


Fig._8

RUTINA DE SERVICIO
AL RELOJ

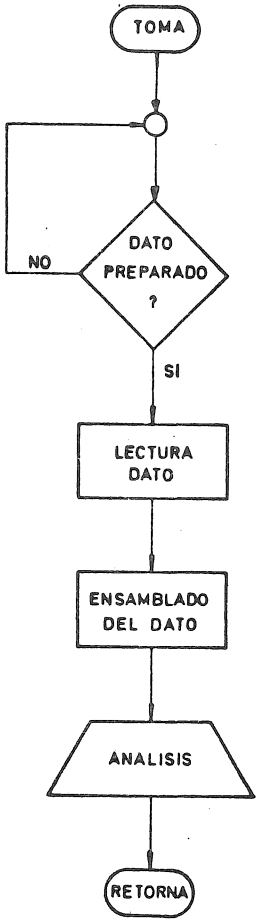
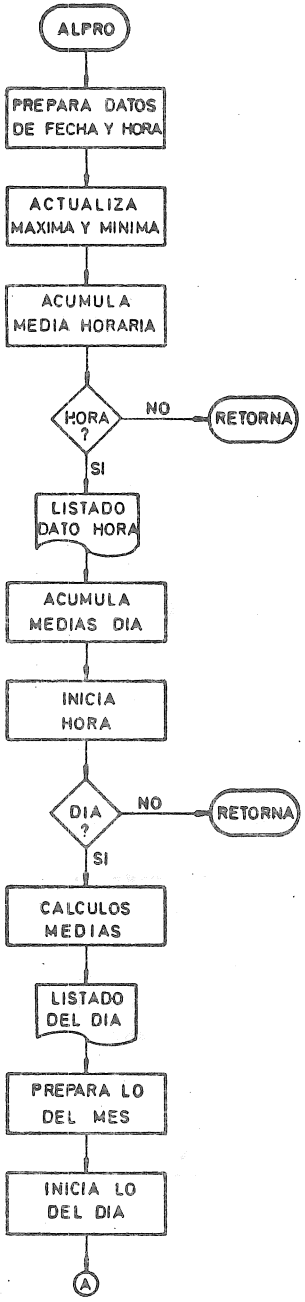


Fig._9

PROGRAMA DE TOMA DE DATOS



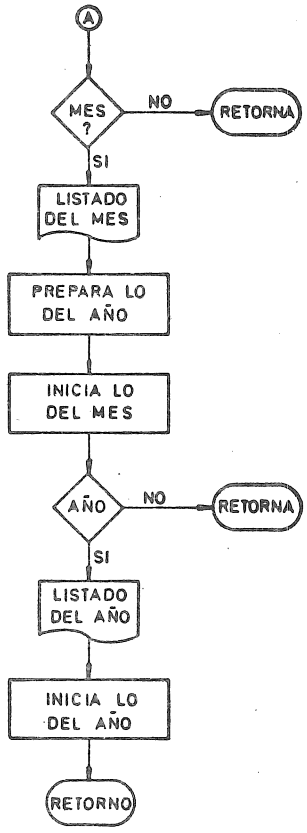


Fig._10
RUTINA DE ANALISIS
Y CALCULOS

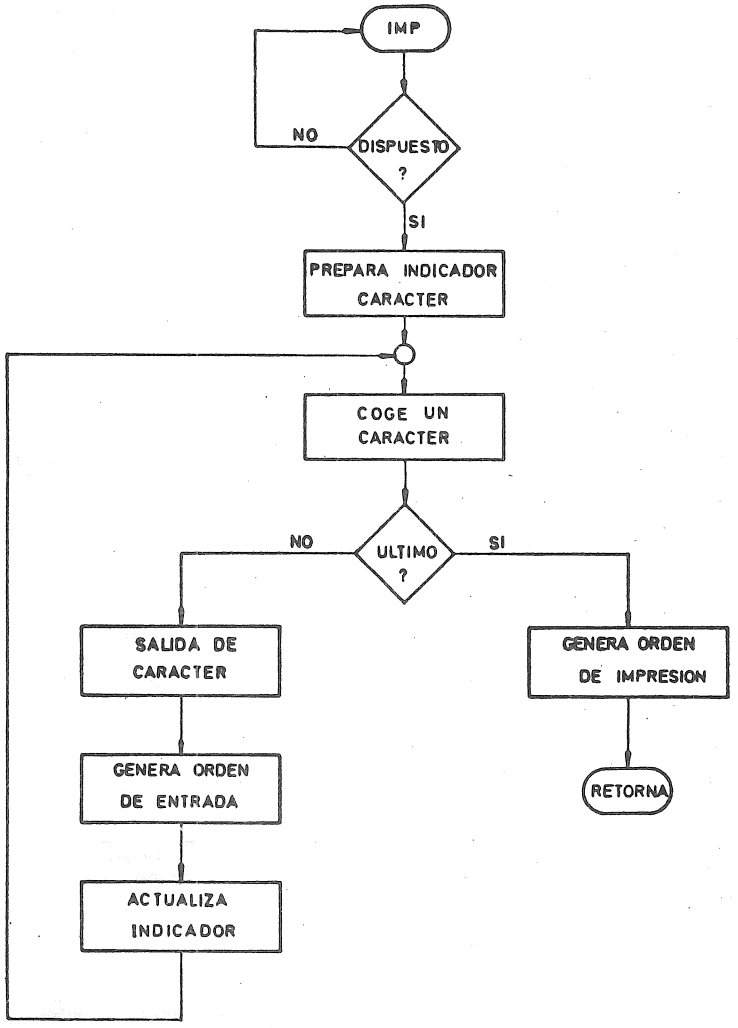


Fig._11
DIAGRAMA BLOQUE DE LA
SUBRUTINA DE IMPRESION DE LINEA

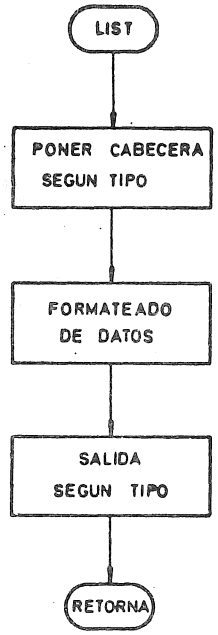


Fig. _ 12

RUTINA DE LISTADOS

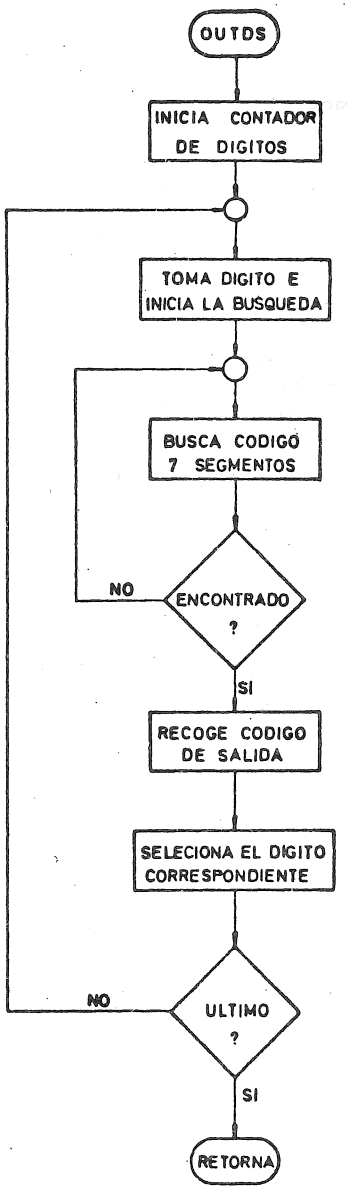


Fig. _ 13

RUTINA DE REFRESCO DEL VISUALIZADOR

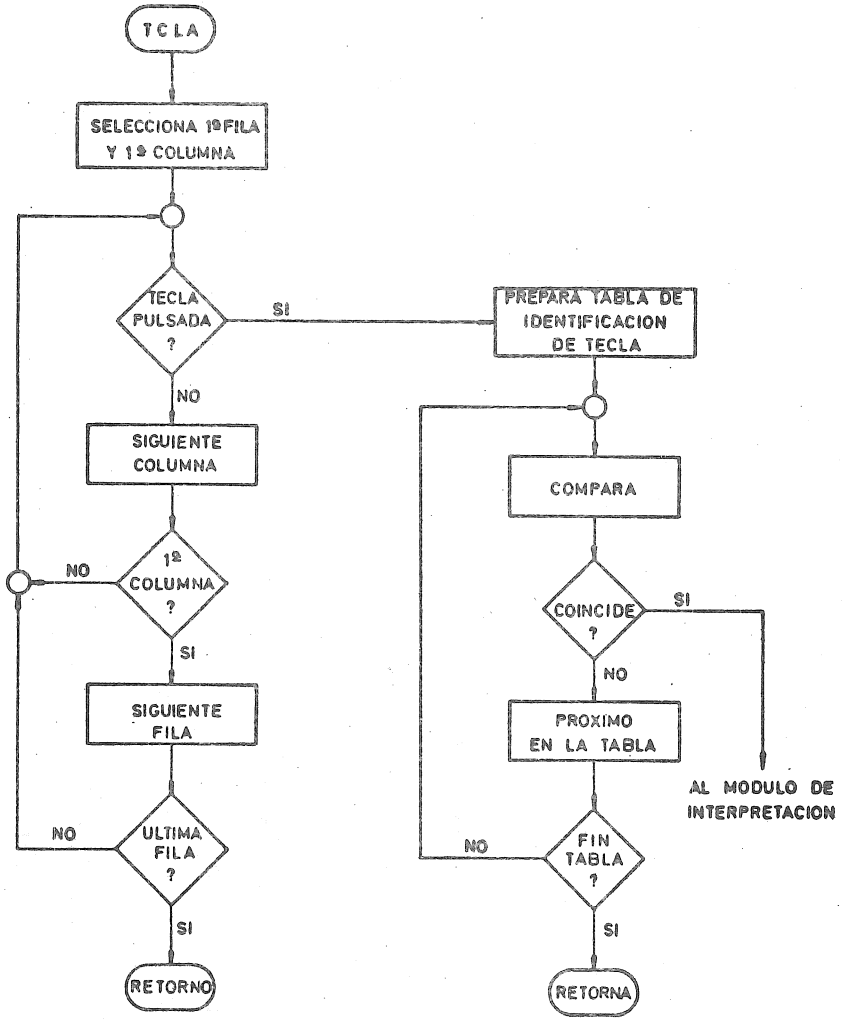


Fig._ 14

RUTINA DE ESCRUTADO DEL TECLADO

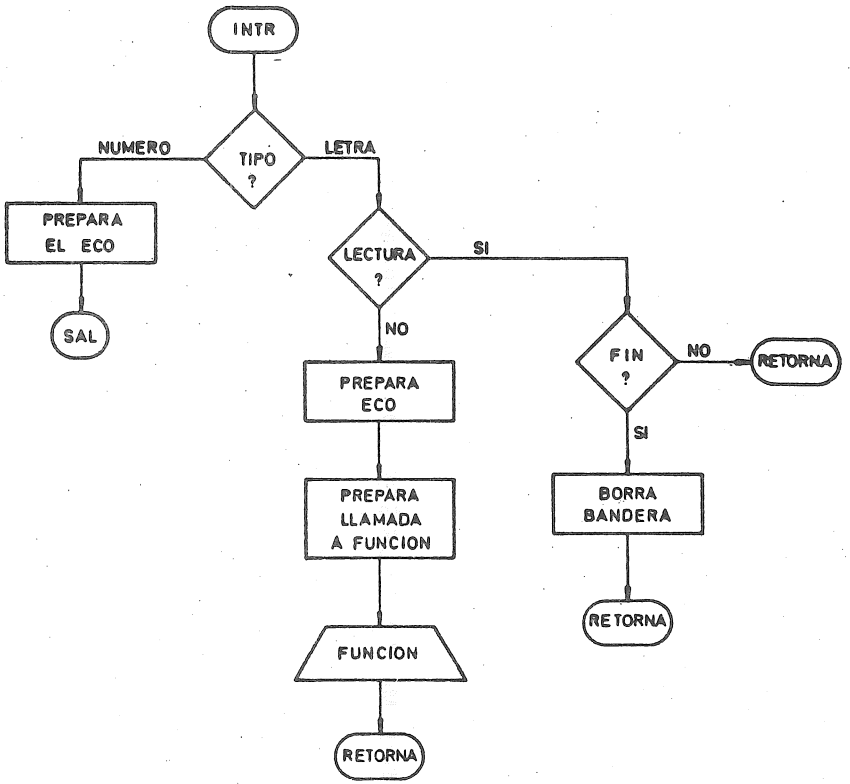


Fig._15

RUTINA INTERPRETE DE TECLA

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

DISEÑO AUTOMATIZADO: METODO HEURISTICO DE PARTICIONAMIENTO

Daniel H. Magni

Osvaldo E. Agamennoni

Rafael O. Fontao

Universidad Nacional del Sur
Bahia Blanca, Argentina

INTRODUCCION

Uno de los problemas que encuentra un programador al trabajar con memorias divididas en páginas y su programa ocupó varias páginas, es el de distribuir las instrucciones, o las subrutinas, o grupos de instrucciones, en las páginas de manera de minimizar las referencias y transferencias de control que estén en páginas diferentes.

Cada una de esas instrucciones o grupo de ellas o subrutinas, puede ser pensado como grupos interconectados por la secuencia del programa o bien las transferencias de control.

La distribución buscada se puede hacer aplicando los métodos de particionamiento de elementos interconectados los cuales minimizan la cantidad de interconexiones entre los subgrafos en que se ha particionado.

Otro gran campo de aplicación, que en realidad dio origen al método

que aquí se presenta, es en el diseño de tarjetas de circuitos impresos. Esto es, cuando la cantidad de componentes electrónicos que constituyen el circuito eléctrico excede la capacidad de cada tarjeta, es aquí donde debemos particionar, minimizando la cantidad de conexiones entre tarjetas.

METODOS DE RESOLUCION

Existen dos maneras principales de resolver este tipo de problemas y ellas son óptimas y subóptimamente.

La primera es factible sólo cuando la cantidad de elementos a particionar es muy pequeña, ya que el método a usar deberá ensayar todas las soluciones posibles, quedándose con la mejor. Pensemos que la cantidad de elementos es por ejemplo 40 y se quiere particionar en 4 grupos de 10 elementos cada uno; la cantidad de soluciones a ensayar es mayor que 10 elevado a la potencia 20!!, lo que representa un tiempo de computación estraorfinariamente grande.

En contraposición los métodos subóptimos se caracterizan por llegar muy rápidamente a soluciones que se acercan a la ideal y en algunos casos la obtienen.

Dentro de estos métodos nos encontramos con constructivos e iterativos.

Los métodos constructivos, como su nombre lo indica, construyen una solución y se basan en agrupar los elementos que están muy interconectados entre sí en "cluster" o grupos.

Existen dentro de los constructivos, dos procedimientos distintos y que son los secuenciales y paralelo. La experiencia indica que usando el procedimiento secuencial, el cluster esencial está muy conectado pero los siguientes no lo están tanto.

En el paralelo se hace difícil seleccionar las semillas o elementos que dan origen a los clusters, ya que la misma se hace previo al agrupamiento.

Ambos procedimentos pueden ser pensados como yn caso especial de un procedimiento más genral que genera clusters de E elementos para un "semillero" inicial y luego genera clusters de E' elementos con los que no han sido asignados y hace corresponder a c/ semilla un grupo de estos.

Lo que generalmente se hace, es combinar ambos métodos (serie y paralelo) para reunir ventajas de cada uno.

Mucho más sencillo se presentan los métodos iterarivos de partición, o distribución inicial y lo van mejorando iterativamente. Un ejemplo de estos métodos es el de KERNIGHAM - LIN (7).

DEFINICION DEL PROBLEMA

Dado un conjunto de n nodos interconectados, cuyas interconexciones están definidas por una matriz de conexiones C(I,J), donde c_{ij} son las conexiones entre los nodos i y j, y grupos de un tamaño determinado, el objetivo es particionar los n-nodos en g- grupos de tal manera que se minimicen las conexiones entre los g- grupos.

La matriz de conexiones es simétrica, de diagonal principal nula. La función minimizar es:

$$F = \sum_{\substack{i=1 \\ j=i+1}}^{n} C_{ij} \quad \text{, tal que i y j estén en distintos grupos}$$

PRESENTACION DEL METODO

Ubicamos nuestro método dentro de los subótimos iterativos.

Parte de: la matriz de conexiones C_{ij}, de la matriz de distancias $D(P_i,P_j)$, un lugar virtual y ovbiamente de una partición inicial. La utilización de la matriz de distancia se explicará más adelante.

Como función a minimizar tenemos entonces:

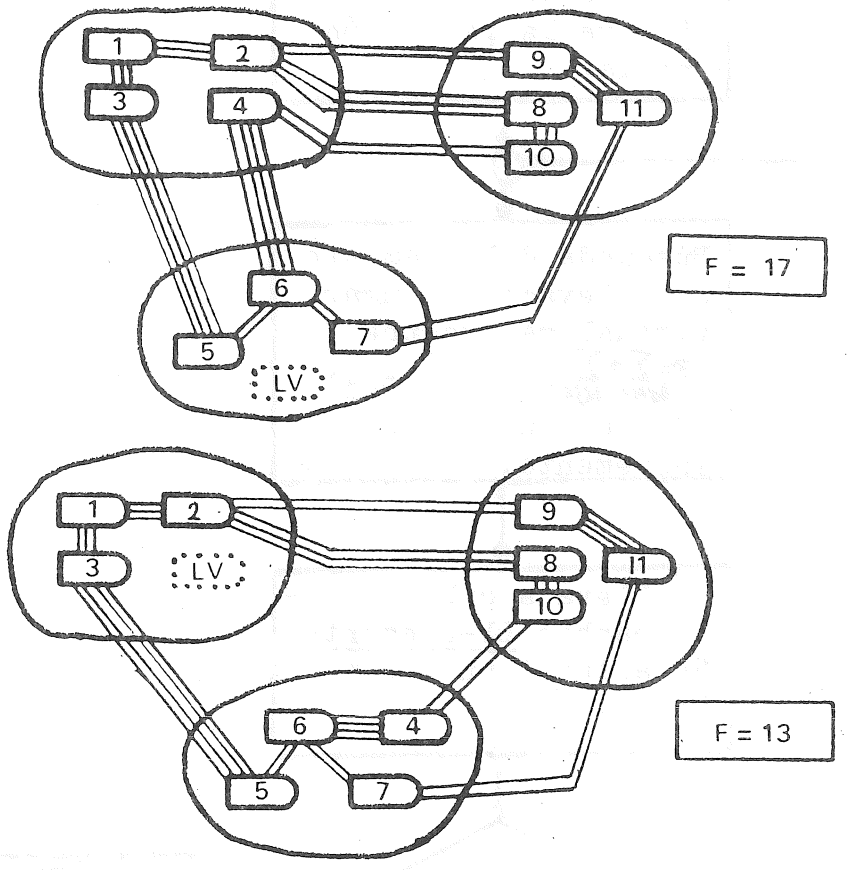
$$F = \sum_{1 \leq i \leq N} \; \sum_{1 \leq j \leq i} C_{ij} \times D_{P(i)P(j)}$$

siendo N = cantidad de elemen-
P(i), P(j) posición del elemento i, j respectivamente.

El método comprende dos fases:

1- La primera fase consta de N-iteraciones donde en cada una de ellas el algoritmo busca cual de todos los elementos de los subgrafos en que no está el lugar virtual (L.V.), tal que llevado al subgrafo donde se encuentra el L.V., reduce más la función a minimizar F, (en ca-

so de no encontrar ninguno, se queda con el que menos la incremente) y efectúa el intercambio de elemento - lugar virtual.

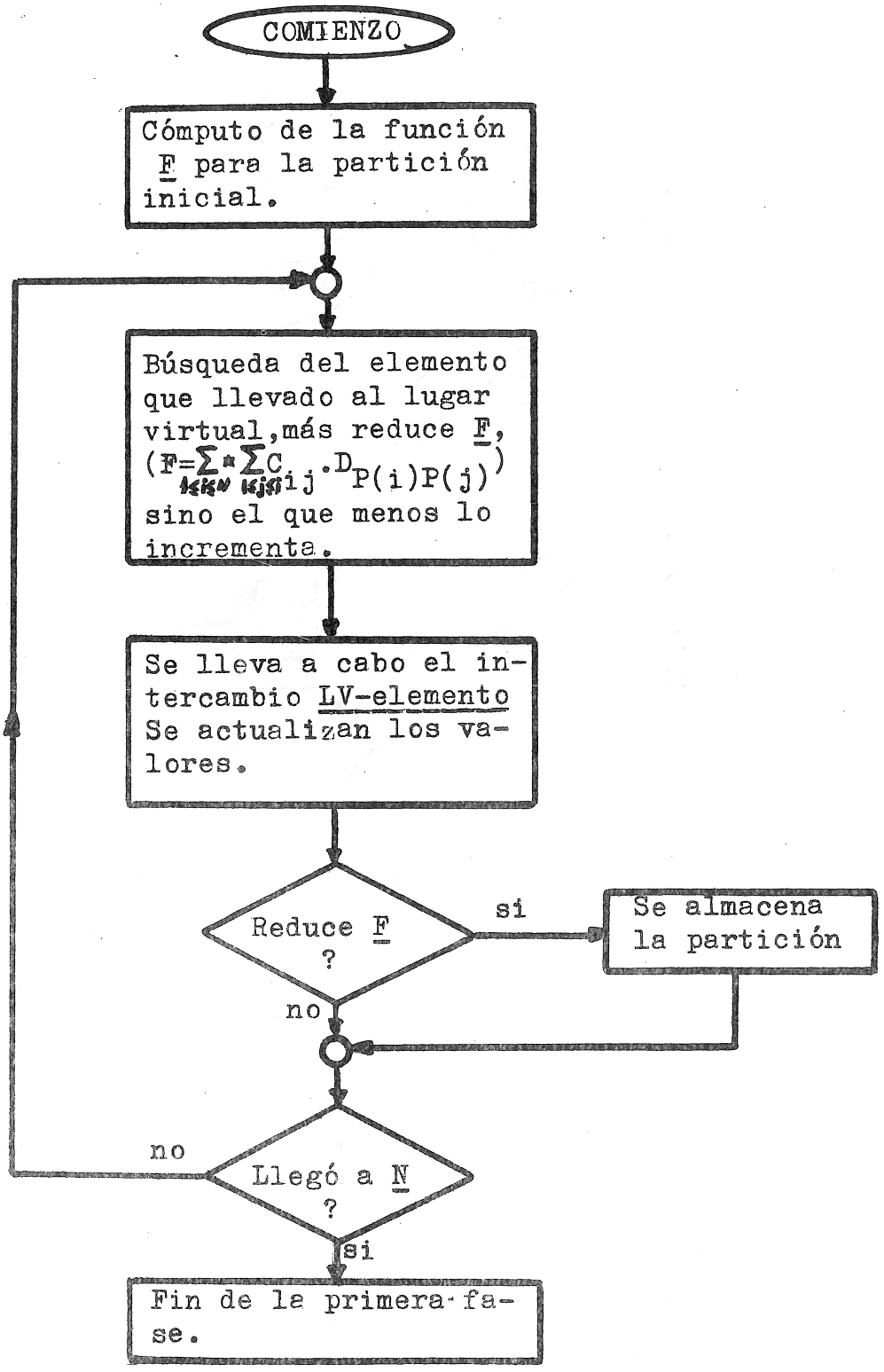


EJ DE UNA ITERACION

El objetivo de usar la matriz de distancia es principalmente el de hacer particiones teniendo en cuenta la distancia física a que se encuentran los subgrafos.

Para el cómputo de la función F a minimizar, los elementos que estén en el mismo subgrafo tendrán distancia nula y solo aportan a F los que se conectan con otros subgrafos.

Si todos están a la misma distancia, la matriz D tomará el valor uno y solo aparta a F las conexiones entre subgrafos. A continuación se muestra un diagrama bloque de la fase 1


COMIENZO
Cómputo de la función F para la partición inicial.
Búsqueda del elemento que llevado al lugar virtual, más reduce F, ($F=\sum_{1\le k\le N}\sum_{k\le i\le j}C_{ij}.D_{P(i)P(j)}$) sino el que menos lo incrementa.
Se lleva a cabo el intercambio LV-elemento Se actualizan los valores.
Reduce F ?
si
Se almacena la partición
no
Llegó a N ?
no
si
Fin de la primera fase.

Es posible que la partición final resulte tal que en uno de los subgrupos tengo un elemento más con respecto de la partición inicial. Si la cantidad de elementos por subgrupos es estricta, basta imponer como condición que la partción final sea una de las que el LV está en el mismo subgrupo de partida.

2- La segunda fase es la que dispone la partición final de alguna forma tal que aplicada nuevamente al algoritmo, éste llegue a particiones de menor cantidad de interconexiones. La experiencia indica que produciendo un movimiento o sacudón dentro del sistema, este es llevado a otro estado o partición que no haya sido obtenida por el método y tomarla como partición inicial para aplicar nuevamente el método.

CONCLUSION

Frente a otros métodos iterativos de particionamiento, el comportamiento de éste procedimiento se presenta mucho más versátil, puesto que particiona con igual sencillez en grupos de 2, 3,...K; por ejemplo KERNIGHAM, B. W.: LIN, S. básicamente particiona en 2 grupos, si se quiere más grupos la mecánica es 1° dividir en 2; cada uno de los que resultan, nuevamente en 2 y así sucesivamente.

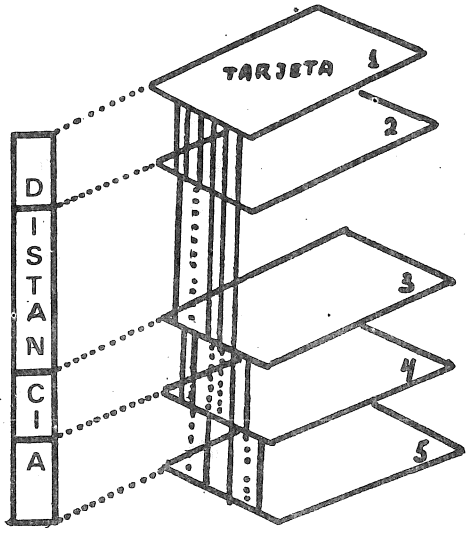
Se puede apreciar que si no son pares, hay que recurrir a artilugios como el de crear elementos ficticios.

Es sumamente veloz, siendo su complejidad de aproximadamente al cuadrado de los elementos.

No importa el tamaño de cada grupo y lo que consideramos de mayor utilidad es el hecho de poder considerar o reflejar la distancia física a quese van a encontrar los subgrupos.

Con esta última característica fue usado con resultados ampliamente satisfactorios, para particionar 200 componentes de un circuito electrónico en 5 grupos o tarjetas (donde finalmente se dispusieron trazándose el circuito impreso de la misma) que debían estar una encima de la otra y a una distancia dada.

La figura muestra un esquema del ejemplo.



BIBLIOGRAFIA

1. AGAMENNONI, O.; MAGNI, D.; FONTAO, R. :"Diseño Automatizado: Método Heurístico de Colocación" VII Conferencia Latinoamerica de Informática. Caracas 1980. Enero.

2. BENTLEY, J. L.; FRIEDMAN, J. H. : "Fast Algorithms for Consturcting Minimal Spanning Trees in Coordinates Spaces"
IEEE Trans. on Computer. Vol. C-27, Feb. 1978.

3. FORD, L. R. ; FULKERSON, D. R. :"Flows in Network "Princeton University Press, 1962.

4. FRIEDAM, A. D.; MENNON, P. R. : "The ory & Design of Switching Circuits". Computer Science Press, Inc. 1975

5. GAREY, M. R.; HWANG, F. K.; JOHSON, D. S. : "Algorithms for a set Particioning Problem Arising in the Design of Multipurpose Units". IEEE Trans. on Computer Vol. C-26, N°4- April 1977.

6. GILBERT, E. N.; POLLAD, H. O. : "Steiner Minimal Trees"
SIAM J. Appl. Math.; Vol 16, N°1 1968.

7. KERNIGHAM, B. W.: LIN, S. : "An Efficient Heuristic Procedure for Particioning Graphs"- The Bell System Technical Journal. Feb 1970.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

DISEÑO AUTOMATIZADO: COLOCACION CON TAMAÑOS DISIMILES

Daniel H. Magni

Universidad Nacional del Sur
Bahia Blanca, Argentina

INTRODUCCION

Dentro de la problemática del diseño de circuitos impresos asistido por computadora, uno de los tópicos importantes, es el de distribuir o "colocar" los componentes de una manera óptima sobre la tarjeta.

Muchos son los parámetros que se deben tener en cuenta para hacer una buena colocación: térmicos; físicos, eléctricos, distancia entre componentes, etc.

Elegir un método que optimice tantos parámetros a la vez, resulta costoso desde el punto de vista de tiempo de computación y complejidad.

La idea es buscar métodos prácticos y fundamentalmente rápidos. Es por esta razón que en vez de tantos parámetros, se busca optimizar uno de ellos que, en cierta forma, englobe a los demás y que como método no llegue a la solución ideal sino que, muy rápidamente, se acerque a la misma e incluso puede llegar a ella.

Un método de colocación con estos requisitos es el que

presentamos en la VII Conferencia Latinoamericana de Informática:"Diseño Automatizado:Método Heurístico de Colocación" y el parámetro que minimiza y que en general la mayoría de los métodos de colocación usan es el de la longitud total de interconexiones.

El mismo era para resolver problemas donde se consideraban todos de igual tamaño.

En este artículo se presenta una metodología para resolver el problema de colocación cuando difieren los tamaños de los componentes.

En la misma se usan técnicas de particionamiento y colocación, sobre la base de la aplicación de un único algoritmo.

DESCRIPCION DEL METODO

En la metodología a seguir para efectuar una colocación óptima de componentes de distinto tamaño, se pueden distinguir 2 etapas principales. Las mismas pueden resumirse en una de particionamiento y otra de colocación.

Se describirán a continuación las mismas junto con lo que sería la resolución de un pequeño ejemplo en el cual la colocación que se debe hacer incluye componentes de 2 tamaños el cual en este caso, el tamaño grande equivale a 3 de los chicos.

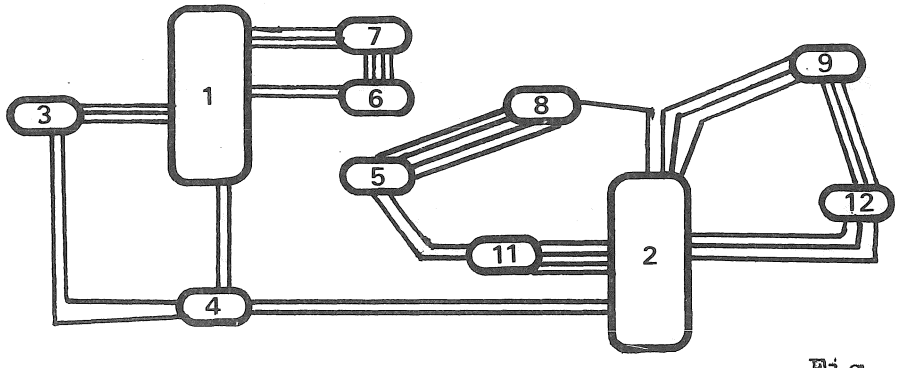


Fig. 1

La primera etapa consiste de un particionamiento. Con la matriz de conexiones $C_{i,j}$ (donde $c_{i,j}$ son las conexiones entre los elementos i y j), como datos del problema, se forman grupos que van a estar constituídos por un elemento grande o bien por tres chicos.

Estos grupos son la partición inicial que se mejorará iterativamente tratando de minimizar la cantidad de conexiones entre grupos.

Aquí los elementos que se puedan mover para mejorar la partición, son los chicos ya que como los grupos son de igual tamaño, donde se encuentra el grande está completo.

La figura muestra una partición inicial:

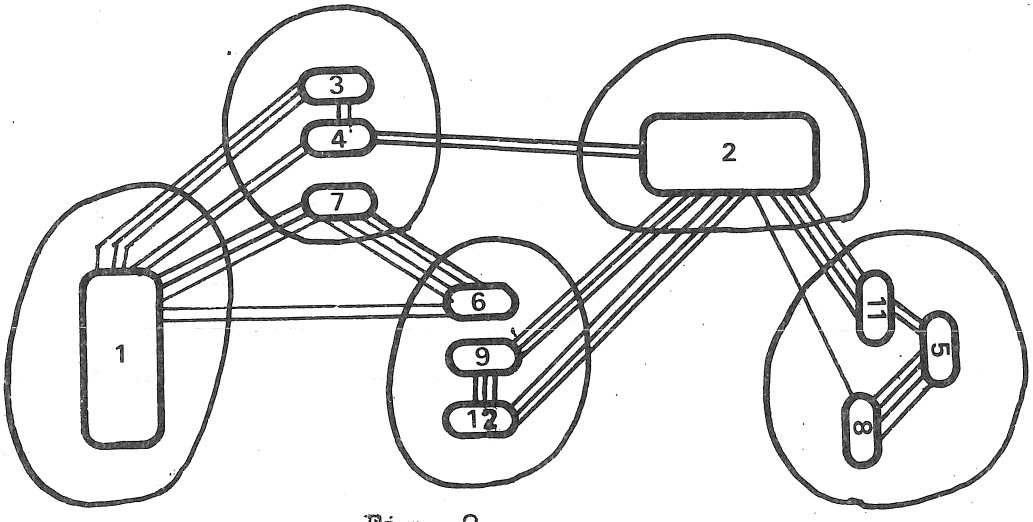


Fig. 2

La siguiente etapa posiciona los grupos resultantes dentro del reticulado en que se ha dividido la tarjeta en que se dispondrán.

Esta sería la colocación inicial que el algoritmo de "colocación" mejorará iterativamente minimizando la longitud total del interconexionado.

Para el algoritmo cada uno de los grupos sea de uno o tres elementos, es indiferente ya que los toma como nodos.

La colocación resultante da una idea acabada de la posición de los elementos grandes sobre la tarjeta. Los elementos chicos todavía no son tenidos en cuenta individualmente sino como grupos de a tres.

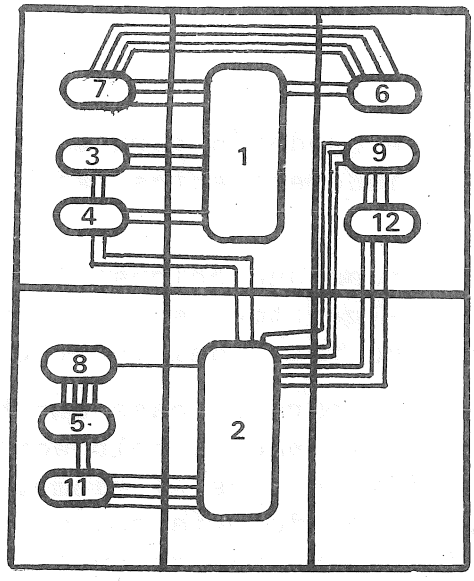


Figura 3

En la colocación final, los elementos grandes se inmovilizan o fijan en la retícula donde quedaron posicionados.

Una vez fijados los elementos, lo que se hace es un cambio de reticulado, llevando el mismo a un paso de retícula en el que entra solamente un componente chico.

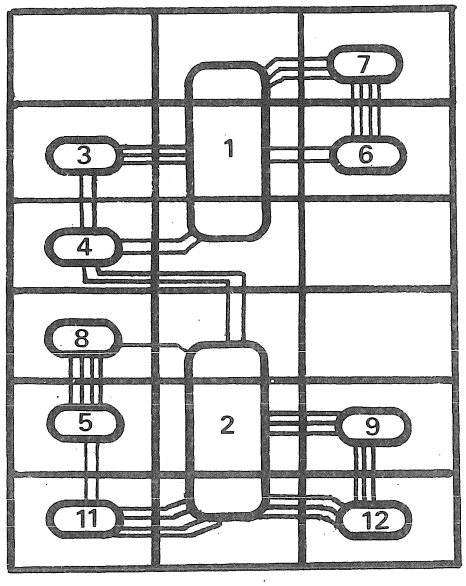


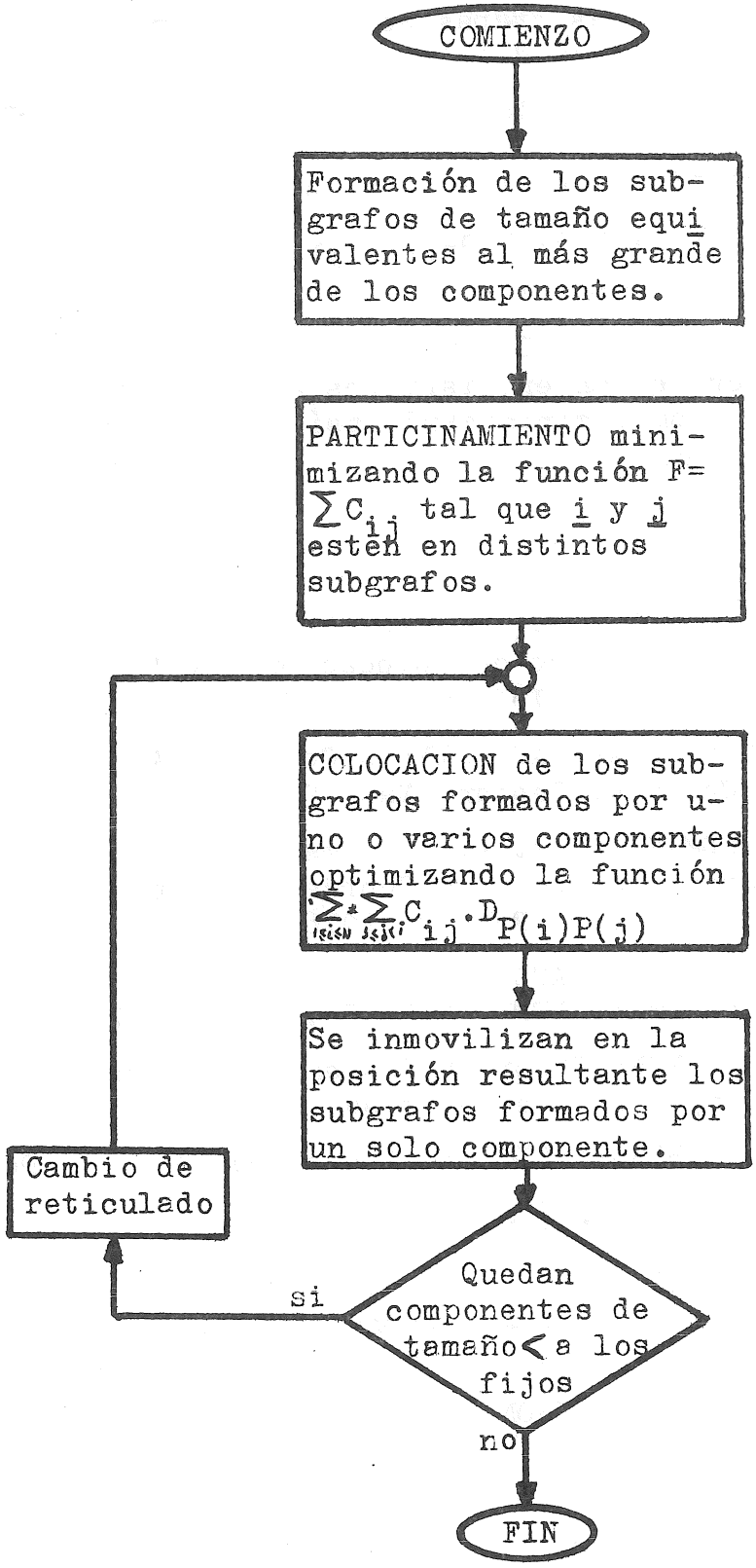
Figura 4

Los elementos grandes quedarán para el algoritmo como si fueran tres elementos independientes cada uno de los cuales es inamovible.

Se lleva a cabo entonces una colocación de los componentes chicos, con el mismo algoritmo del paso anterior, es decir minimizando la longitud total del interconexionado.

Hasta aquí se presentó para el caso que hubieran dos tamaños solamente. Se puede extender a colocaciones donde haya varios tamaños. El procedimiento es el mismo, salvo que se repiten algunos pasos. La metodología sería la siguiente: particionar en grupos de tamaño equivalente al más grande. Colocación y fijación de esos elementos. Cambio de reticulado y colocación del tamaño que sigue y fijación del mismo. Cambio del reticulado, colocación y fijación del que sigue y así sucesivamente hasta el elemento más chico.

El diagrama bloque de la página siguiente ilustra la metodología.


COMIENZO
Formación de los subgrafos de tamaño equivalentes al más grande de los componentes.
PARTICINAMIENTO minimizando la función F= $\sum C_{ij}$ tal que i y j estén en distintos subgrafos.
COLOCACION de los subgrafos formados por uno o varios componentes optimizando la función $\sum_{i \leq i \leq N}^{*} \sum_{j \leq j \leq i} C_{ij} \cdot D_{P(i)P(j)}$
Se inmovilizan en la posición resultante los subgrafos formados por un solo componente.
Quedan componentes de tamaño < a los fijos
si
Cambio de reticulado
no
FIN

ALGORITMOS

Esto que parece un procedimiento tedioso o largo, es en realidad, sumamente sencillo, práctico y fundamentalmente rápido ya que en todas las etapas se usa un mismo algoritmo de optimización descripto en detalle en la referencia (1).

Básicamente minimiza la función $F=\sum_{1\leq i\leq N} * \sum_{1\leq j\leq i} C_{ij} D_{P(i)P(j)}$

donde N es la cantidad de elementos, c_{ij} son las conexiones entre los elementos i y j, y $d_{P(i)p(j)}$ es la distancia entre las posiciones que ocupan i y j.

En particionamiento, D toma el valor de 1(uno) para aquellos elementos que estén en distintos grupos y cero para los que esten en el mismo, minimizando así la cantidad de conexiones entre grupos.

BIBLIOGRAFIA

1. AGAMENNONI, O;MAGNI, D.; FONTAO, R.: "Diseño automatizado:Método Heurístico de Colocación". VII Conferencia Latinoamericana de Informática. Caracas. Enero 1980.

2. AAKHUS, SEEMAN, PTAK:"ACCLAIM: A Computer Aided Design System". Computer Design. May 1968.

3. BENTLEY, FRIEDMAN:" Fast Algorithms For Constructing Minimal Spanning Trees in Coordenates Spaces". IEEE Trans. on Computer. Febrero 1978.

4. FISK, CASKEY, WEST:" ACCEL- Automated Circuit Card etching Layot"- Proc. IEEE, Vol. 55, Nov. 1967.

5. FRIEDMAN and MENON:"Theory & Design of Switching Circuits"- Computer Science Press, Inc. 1975.

6. HANAN and KURTZBERG:" Placement Rechniques"., in Design Automation of Digital Systems:Theory and Techniques- Ed. M.A. Breuer. New York: Prentice Hall, 1972,ch.5.

7. HANAN, WOLFF, and AGULE:" A Study of Placement Techniques for Computer Logic Graphs"- Proc. 13 th Design Automation Conf., June 1976, pp. 214-224.

8. GAREY, HWANG, and JOHNSON:" Algorithms for a Set Partitioning Problem Arising in the Desing of Multipurpose Units". IEEE Trans on Computers, Vol, C-26, N° 4- Apr, 1977.

9. HIGHTOWER, D.:"The Interconnection Problem:A Tutorial" IEEE Computer Mazazine, Vol. 7, N° 4- Apr, 1974.

10. GOTC, and KUH:" And Approach to the Two-Dimensional Placement Problem in Circuit Layout"- IEEE Trans. on Circuits and Systems, Vol, CAS-25,N° 4- Apr. 1978.

11. KERNIHGAN and LIN:"An Efficient Heuristic Procedure for Partitioning Graphs"- The Bell System Technical Journal- Feb. 1970.

12. PERRILL, W.A.:"Packaging Printed Circuit Boards With Interactive Graphics"- Pergamon Press, Computer & Graphics, Vol 2- 1977.

13. RUTMAN, R.A.:" An Algorithm for Placement of Interconnnected Elements Based on Minimun Wire Length "- Proc. 1964 SICC, pp 477-491.

14. STEINBERG, L.:"The Backboard Wiring Problem:A Placement Algorithm"- SIAM Review, Vol, 3, N° 1, pp 37-50,Jan. 1961.

15. STEVENS, J.E.:"Fast Heuristic Techniques for Placing and Wiring Printed Circuit Boards"- Ph. D Thesis, Computer Science Dept. University of Illinois, Urbana,1972

16. HALL, K.M.:" An R-Dimensional Quadratic Placement Algorithm"- Management Sci., Vol, 17, pp 219-229 Nov.1970

17. QUINN and BREUER:" A Forced Directed Component Placement Procedure for Printed Circuit Boards"- IEEE Trans, on Circuits and Systems, Vol. CAS- 26,N° 6. June 1979.

18. SPITALNY and GOLDBERG:" ON-Line Graphics Applied to Layout Design of Integrated Circuits". Proc. of IEEE, Vol.55, N°11- Nov. 1967.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

DESARROLLO DE UN COMPUTADOR PARA MEDIR VELOCIDAD AEREA VERDADERA

José Schlein

Laboratorio de Electrónica Computacional de la
Facultad de Tecnología de la Universidad de Belgrano
Buenos Aires, Argentina

Este trabajo fué auspiciado por el Departamento de Investigación y Desarrollo de la Fuerza Aérea Argentina

MEDICION DE LAS PRESIONES Y TEMPERATURAS EN UN AVION EN MOVIMIENTO

En general, los aviones actuales vienen provistos de una "antena" Pitot, es decir, un tubo de Pitot colocado en un lugar donde las perturbaciones provocadas por el desplazamiento del avión en el aire sean mínimas.

De la antena Pitot se obtiene dos señales de presión

a) Presión total P_t (en el orificio central)

b) " estática P_s (en los orificios laterales)

De las relaciones

$$\frac{P_t}{P_s} = (1 + 0,2\ M_t^2)^{3,5}$$

y $$M_t = \frac{V_t}{Cs} = \frac{\text{Velocidad aérea verdadera}}{\text{Velocidad del sonido}}$$

Se puede obtener:

$$V_t = 39,84 \quad \frac{T_i}{1/M_t^2 + K_r}$$

Donde T_i = temperatura medida en un sensor de tempratura con coeficiente K_r de recuperación.

Para poder computar el valor de V_t es necesario medir P_s, P_t y T_i, con una precisión acorde al resultado que se quiera obtener.

Por ejemplo: para obtener V_t con un error menor que el 0,5%, será necesario tener los valores de P_s, con un error menor del .1% P_t menor del .1% y T_i menor del 0,3%.

Los valores del número de Mach determinados a partir de esos valores de presión se llaman de "Mach indicado".

Cada avión tiene, para el conjunto formado por el fuselaje y el tubo de Pitot, una curva de corrección del nº de Mach debido a los errores provocados por las perturbaciones del propio avión.

Para el A4B, por ejemplo, se cuenta con la curva que convierte el Mach indicado a Mach real.

Existe además una curva que permite corregir el valor de la presión estática por las perturbaciones, en función del valor de velocidad indicada con la cual se puede corregir el valor de altura calculada con el valor de P_s obtenido del tubo de Pitot.

Para sensar las presiones con una exactitud suficiente (del órden del 0.1%) se eligen los sensores PAROSCIENTIFIC que consisten en un oscilador a cristal de cuarzo cuya frecuencia varía con la tensión a la cual está sometido, que a su vez está relacionada con la presión (absoluta ó diferencial) que mide (1). Cada sensor particular esta calibrado para tener los valores de A, B, y T_o

$$(1) \quad P = A\left(1 - \frac{T_o}{T}\right) - B\left(1 - \frac{T_o}{T}\right)^2$$

T = período de oscilación con la presión P

A, B, y T_o = coeficientes de calibración

Es decir que la medición de presión se convierte en una medición del período de oscilación.

Dado que la frecuencia es la inversa del período, también

$$P = A\left(1 - \frac{F}{F_o}\right) - B\left(1 - \frac{F}{F_o}\right)^2$$

que tambien es una forma válida de medir la presión. Ambos métodos se ilustran en Fig.1, habiéndose elegido el de medición de período para este proyecto.

La medición de frecuencia o período debe dar una precisión y resolución suficientes para que P tenga asimismo precisión y resolución adecuadas. Suponiendo a A, B, y T_o con errores desprecialbes, para medir P con .1% de error a plena escala sería necesario un error menor del .01% puesto que por el rango de variación de la frecuencia de salida (de 36 a 40K Hz) para una variación de la presión de entrada de 0 a plena escala (100%) solo se varía un 10% la frecuencia de salida.

Esto determinaría que fuese necesario medir el período con $\pm$ .01%, es decir midiendo 10.000 pulsos de la frecuencia de conteo (por ejemplo 1M Hz).

Pero con esa cantidad, la resolución de la ecuación (1) hace que para bajas presiones no haya suficiente resolución (de diferencia de dos valores posibles de presión consecutivos mayor del .1%).

Eso nos hace imprescindible aumentar la cantidad de pulsos a contar por dos métodos 1) Aumentar la frecuencia del reloj patrón; 2) Contar mas pulsos de salida del sensor.

Se decide que, dado que existe un cristal en el equipo que genera una frecuencia de 1M Hz, se lo puede usar como patrón de tiempo y será entonces necesario incrementar el número de períodos de señal durante el cual contar pulsos.

Para obtener un adecuado nº de pulsos es necesario contar 64.000 razón por la cual obrendríamos números de 16 bits con contadores acordes.

Los contadores comerciales cuyas salidas son accesibles son los del tipo 4040 (de 12 bits) y necesitamos 16 bits de resolución, dado que esos don los bits necesarios para que, por ejemplo a Mach 0,8 y Hp = 40.000 pies tengamos un error en TAS menor de 0,5%, especificación del sistema. Este problema se encuentra cuando las presiones a medir son bajas, como en el caso mencionado.

Podemos adoptar como solución acoplar otro contador más (por ejemplo un 4024 de 7 bits) pero como los números de salida van a estar comprendidos entre CXXX y DXXX, se puede prescindir de los tres primeros bits (mas significativos) colocando en el registro de salida esos datos fijos en forma permanente. Queda todavía un bit por definir. Ese bit puede ser el bit 0 ó el bit 12 según se coloque un divisor detrás del 4040 ó un flip-flop delante, con la ventaja que, colocando ese flip-flop se puede reducir el tiempo de conversión a la mitad del que sería si hubiese un divisor posterior.

Se eligen entonces para contar los pulsos del sensor la siguiente configuración

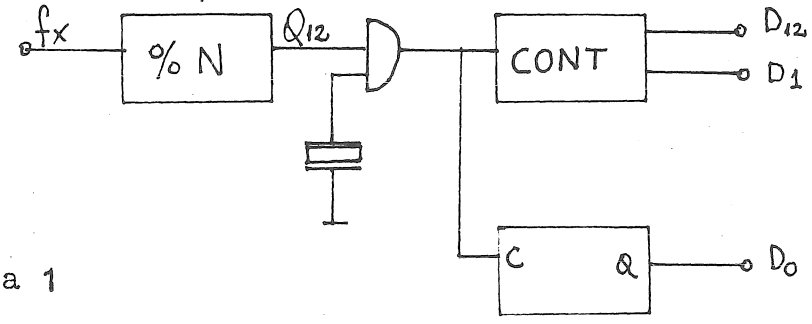


Figura 1

De esta manera se obtiene los 16 bits de información en 58 mseg aproximadamente, con números comprendidos entre $ CC60 y $ DF 50 para el par de sensores actualmente en nuestro poder y para el rango de presiones a que deberán estar sometidos.

(P_s = 75 a 800 mbar; Pt - Ps = Qc = 0 a 800 mbar dif)

Por ejemplo, para el sensor P_s tenemos

A = 11486 milibares
B = 6445,3 "
T_0 = 25,37936 Seg

La relación entre P_s y T es:

$$T = T_0 \frac{2B - A - \sqrt{A^2 - 4BP_s}}{2 (P_s - A + B)}$$

Para P = 75 mbar T = nº 52320, $ CC60
P = 800 " T = $ DAEE

Para el sensor de presión diferencial:
A = 10981,6
B = 5999,77
T_0 = 25,79242

Para Q_c = 0 $ C E 57
Q_c = 800 $ D F 50

Tenemos el rango del nº que representa la presión será

CC 60 1 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0
DF 50 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0

Los tres primeros bits son fijos, por lo que solo será necesario discriminar los trece restantes con el contador de 12 bits tipo 14040 y el flip-flop D tipo 4013.

Para poder efectuar, una vez obtenido el número, el periodo del sensor para poder calcular la presión se debe convertir el número obtenido al periodo real.

La compuerta de la figura 1 dejará pasar los pulsos del relój patrón mientras la salida del divisor por N esté en su estado alto, es decir durante medio periodo de la onda de salida.

Si tomamos como salida Q_{12} del 4040, el periodo se multiplica por $2^{12} = 4096$, es decir que el medio periodo multiplica al periodo original por 2048.

Durante ese tiempo se pueden medir un dado número de pulsos del relój patrón de T pat =1useg, número que será el de salida (N_p) de donde $N_p \times Tpat = T \times 2048$.

$$T = \frac{N_p \times Tpat}{2048} = \frac{N_p \times 1\ useg}{2048}$$

Reemplazando en la ecuación de P = f (T)

$$(A)\ P = A\left(1 - \frac{To}{T}\right) - b\left(1 - \frac{To}{T}\right)^2 = (A) \cdot \left(1 - \frac{To \times 2048}{Np \times useg}\right) -$$

$$-(B)\ \left(1 - \frac{To \times 2048}{Np \times 1\ useg}\right)^2$$

Resulta factoreando

$$(B)\ P = H1 + \left(H - \frac{H2}{N1}\right)\frac{1}{N1}$$

$H1 = A - B$ $\qquad$ $H2 = BK^2$

$H = 2BK - AK$ $\qquad$ $K = \frac{To \times 2048}{1\ useg}$

Para el caso del sensor de presión absoluta

$H1 = A - B = 5040,7$ milibares

$K = 51976,93 = To \times 2048$

$H2 = 1,741263007 \times 10^{13} = B \times K^2$

$H = 73006795,88$

Para el de presión diferencial

$H1 = 4981,83$

$K = 52822,88$

$H2 = 1,674089815 \times 10^{13}$

$H = 53770522,50$

A pesar que la ecuación (A) lleva para su resolución 1 operación de multiplicación más que la ecuación (B), se deberá usar la primera, puesto que las constantes que intervienen en la resolución de la segunda son muy grandes para operar con ellas, se obtienen pues, los datos básicos de presión estática y dinámica.

Para la obtención del dato de temperatura se cuenta con un sensor de platino, cuya Ley R = f(T) es

$$R_t = Ro\ (1 + \alpha \left[T - \left(\frac{T}{100}\right)\ \left(\frac{T}{100} - 1\right)\right]\gamma - \beta \left(\frac{T}{100} - 1\right)\ \left(\frac{T}{100}\right)^3 \Big])$$

$$\alpha = .003925 \qquad \beta = \begin{cases} .1 & T < 0ºC \\ 0 & T > 0\ C \end{cases}$$

$$\gamma = 1{,}45$$

T = temp en º C

Para la medición de esa resistencia vamos a usar una configuración tipo puente de Wheatstone

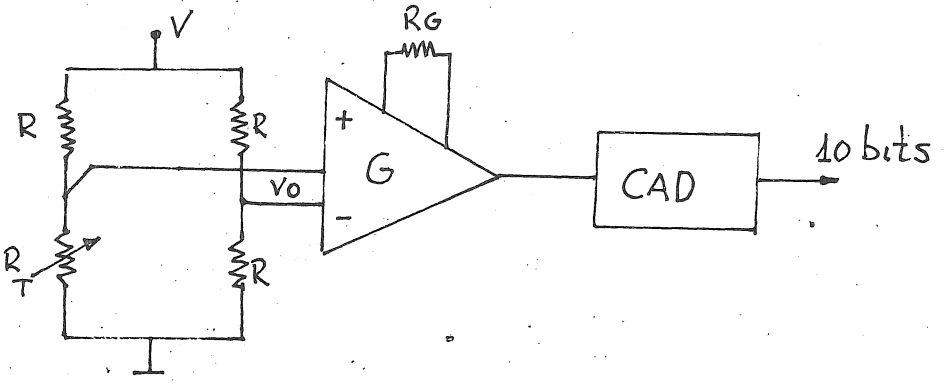


Figura 2

Para que Vo sea de una sola polaridad, R debe ser del valor mínimo de Rt, ó sea el equivalente a una temperatura de -55ºC (389,59Ω.

El valor mas cercano normalizado es de 383 . Dado que Rt a 55ºC (el valor mas alto de temperatura en atmósfera caliente normal) tiene un valor de 608,64Ω y dada la tensión de referencia V = 10V, el rango de variación de tensión Vo será de 0,04V a 1,14V, que es escaso, por lo cual entre el puente de Wheatstone y el conversor analógico-digital que entregará los datos a la computadora deberá haber

un amplificador de instrumentación que lleve esa variación al rango de entrada del conversor.

El CAD elegido es el AD 571, conversor de 10 bits, con registros de salida y rango de entrada 0 a + 10 V.

Por lo tanto, la ganancia G del amplificador será

$$G = \frac{10}{1.14} = \frac{\text{Vin Max}}{\text{Vo Max}} = 8.\emptyset77$$

La ecuación de ganancia del amplificador elegido, el AD 522 es

$$G = 1 + \frac{200.000}{RG} \qquad R_g = \frac{200.000}{G-1}$$

$$= 25733,63$$

(26,1 k) valor normalizado

Con ése valor de R_g, la tensión Vin tendrá un rango de variación entre 0,35 y 9,88 V, con lo que se podrá aprovechar toda la resolución de AD 571, conversor elegido.

Para referencia de tensión usamos el AD 581, cuya precisión alcanza para nuestras necesidades. Dado que debe alimentar al puente de Wheatstone y a los dos conversores digital-analógico (AD7524), se deberá proveer una amplificación de corriente por medio de un transistor PNP. R_s fija el valor a partir del cual comienza a aportar corriente.

Para obtener la correspondencia entre el número que entrega el conversor analógico-digital y la temperatura que mide el sensor se hace una tabla con los valores correspondientes, dado que, como R_t es una función cúbica de la temperatura, es muy difícil realizar la expresión donde se de $T = f(R_t)$. El proceso de cálculo es el siguiente:

$$T \qquad Rt = Ro\left(1 + T - \left(\frac{T}{100}\right)\left(\frac{T}{100} - 1\right) - \left(\frac{T}{100}\right)^3\left(\frac{T}{100} - 1\right)\right)$$

$$Vo = \frac{VREF}{R/RT + 1} \qquad Vi = Vo \cdot G \quad \ldots \quad NT = \frac{1024 \times Vi}{10\ V}$$

Donde finalmente se obtiene $N_t = (T)$ en forma numérica.

Esta tabla es discreta, y se debe interpolar linealmente entre los puntos que figuran en ella.

El proceso de interpolación utiliza la fórmula de la recta que pasa por dos puntos para lograr el valor intermedio, en tanto que las entradas en la tabla se minimizan de tal manera que el error máximo en la linealización sea menor al permitido.

Proceso de cálculo de TAS y H_p

Obtenidos los datos de los respectivos conversores, es decir, NQ_c; NPs, NT, números que representarán a la presión dinámica, estática y temperatura, respectivamente, se comienza con los cálculos

1) Se calcula $$A = \frac{Qc}{PS} = \frac{AQc\ (1 - \frac{Toqc \times 2048}{NQc}) - Bqc\ (\frac{1 - Toqc \times 2048}{NQC})^2}{APS\ (1 - \frac{Tops \times 2048}{NPS}) - Bps\ (\frac{1 - Tops \times 2048}{NPS})^2}$$

Donde AQC; To QC = constantes del sensor diferencial
APS; BPS: TPS " " " absoluto

2) Se calcula $H_p = f(PS)$ de una tabla obtenida de las funciones.

para 76 Ps 169,753 H'p = 40090,76996 - 5665,722379 ln PS ALTA
para 169,753 PS H'p = 44332,30769 1-0,28307401 x PS $\frac{1}{5,256}$

Ambas ecuaciones son las definidas por la atmósfera standard y los valores numéricos dan H'_p en metros con P_s en milímetros de mercurio. Dado que la presión estática medida no está corregida en función del número de Mach o la velocidad, se efectúa una corrección sobre la altura resultante con las tablas que da el manual del avión (AHP en función de la velocidad indicada).

Empíricamente, si el valor A es menor de 0,4 no es necesario corregir pero a partir de ese valor

$$\Delta H_p = -300 \times A + A\ H_p$$

Finalmente la altura resulta

$$Hp = H'p + \Delta H_p$$

3) Por medio de una tabla, dado el valor A sacamos M_i

La tabla la da la función

$$Mi = 5 \quad (A + 1)^{\frac{1}{3,5}} - 1$$

4) Si M_i 0,6, no es necesario corregirlo, pero si M_i 0,6 se lo debe corregir en función de M_i, tabla que dá el manual del avión, para dar M (número de Mach real)

5) Se calcula

$$C = \frac{M^2}{1 + .1995 \times M^2}$$

6) se calcula

$$TAS = 38.94 \quad C \times T$$

7) Se deban sacar los valores de TAS y HP en forma analógica, por lo que se convierten en el AD 7524, con los datos obtenidos. TAS tambien sale con forma digital de 8 bits con un latch open collector TTL.

ESTUDIO DE LOS ERROEES EN EL CALCULO DE LA ALTURA

1) h 11.000 m.

$$h = 44332,30769 - 12549,32411 \; P_s \quad \left(\frac{1}{5,256}\right)$$

El error absoluto máximo que puede admitirse es de alrededor de 25 m.

$$e_{hmáx} = \frac{(E \; P_s)}{5,256} \quad \left(\frac{1}{5,256}\right) + m) \; 12549,3211 \; P_{Smáx} \quad \left(\frac{1}{5,256}\right)$$

Descripción del Diagrama en Bloques

El problema a resolver consiste en calcular la velocidad verdadera del avión y su altura.

Para ello contamos con los siguientes datos:

Pt = Presión total, tomada del tubo de Pitot
Ps = Presión estática, tomada de la toma estática
T = Temperatura, tomada de una toma de temperatura que consiste en una resistencia variable con la temperatura.

Las salidas del sistema son:

TAS ANALOG: Señal analógica proporcional a la velocidad aérea verdadera.

TAS DIGITAL: Señal digital indicando en 8 bits la velocidad aérea verdadera.

h ANALOG: Señal analógica proporcional a la altura del avión.

Interfase para la adquisición de datos

Se eligieron para sensar la presión sensores de cuarso que proporcionan una señal eléctrica cuya frecuencia es proporcional a la presión a medir. La precisión de estos sensores es del orden del 0.1%.

Se diseñó una interfase para medir el período de esta señal tal como será analizado luego. Para la medición de temperatura se utilizó un sistema puente, amplificado luego mediante un amplificador de instrumentación.

CPU

Se eligió pará la CPU un microprocesador NMOS M 6802 de 8 bits.

Este microprocesador incluye 128 bytes de memoria RAM y el circuito de reloj.

MEMORIA PROM

Se diseñó esta memoria con circuitos M 2708 que son EPROM de 1K x 8 y que ocupan un total de 3K que es la longitud del programa.

INTERFASE ANALOGICA DE SALIDA

Se utilizaron conversores Digital/analógicos de tipo CMOS para las salidas de TAS ANALOG y h ANALOG.

INTERFASE TAS DIGITAL

Este circuito provee una señal digital de 8 bits en forma de buffers de colector abierto.

DIAGRAMA EN BLOQUES DEL T.A.C.

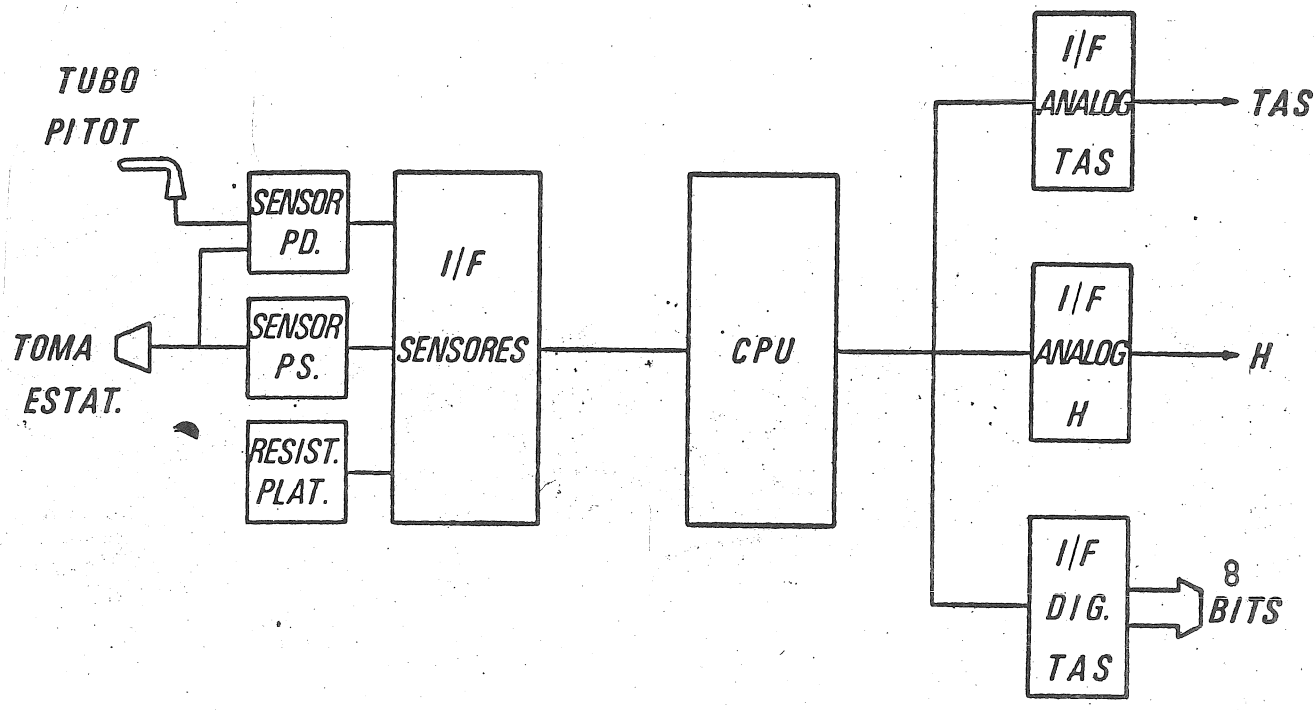


Fig. 8

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

CAPITULO E

ARQUITECTURA DE SISTEMAS Y REDES

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

UNA CENTRAL TELEX CON PROCESADORES TRIPLICADOS

J. A. Grompone

N. M. Mace

Interfase Ltda
Zabala 1372, Montevideo, Uruguay

I. INTRODUCCION

El presente articulo describe el diseño y la realizacion de una Central Telex Automatica, fabricada en Uruguay, basada en un sistema distribuido de computadoras.

Este proyecto, poco usual en un pais en vias de desarrollo, fue el resultado de una Licitacion Publica realizada por ANTEL(1).

Se presentan aqui las caracteristicas tecnicas mas interesantes que resultaron de la experiencia obtenida por las empresas adjudicatarias(2), en el proyecto y la fabricacion de un prototipo de Central Telex. El equipo se encuentra ya en explotacion real y la Administracion ha adquirido otro ejemplar similar.

Ademas de los autores, trabajaron en el proyecto y diseño los ingenieros Juan Gherzi, Jaime Jerusalmi y Enrique Salles.

(1) Administracion Nacional de Telecomunicaciones, empresa estatal que tiene monopolio sobre las telecomunicaciones en el Uruguay.

(2) Las empresa adjudicatarias fueron G.M.S Limitada, Rivera 3314, e INTERFASE Limitada, Zabala 1372, ambas de Montevideo, Uruguay; actuando en forma conjunta para este proyecto.

II. ARQUITECTURA GENERAL

Las caracteristicas generales del diseño fueron establecidas, parcialmente por ANTEL y parcialmente por las empresas adjudicatarias. Se deseaba que el producto final tuviera caracteristicas tecnicas similares a los equipos existentes en el presente. Los puntos mas interesantes que se consideraron fueron:

1. La central estara conectada a la red nacional de telex de 50 baudios, con Alfabeto Telegrafico N.2 (Baudot) [1].

2. La central cumplira con las recomendaciones del CCITT correspondientes a las comunicaciones Telex. [1].

3. Si bien el prototipo se implementara con 128 lineas, se prevera la posible expansion hasta 1024 terminales sin cambios fundamentales en hardware ni programacion.

4. La confiabilidad del sistema es el objetivo principal del diseño y para cumplir con este fin se empleara toda la redundancia necesaria para lograr un tiempo medio entre fallas (calculado) de 10 años.

Se decidio implementar la Central Telex con un sistema distribuido de computadoras que trabajan en tiempo real. Las computadoras poseen dos funciones diferentes: procesadores centrales (o centrales como se dira por brevedad) y procesadores frontales (o perifericos).

La operacion de la Central consiste en la comunicacion de dos grupos de maquinas que intercambian informacion. De las centrales hacia los perifericos se envian ordenes y caracteres Baudot, de los perifericos hacia las centrales se envian mensajes y caracteres Baudot, ver fig. 1.

Los procesadores perifericos se encargan de todo el manejo de las lineas de la central. La informacion serie que llega por cada linea es convertida en un octeto y sincronizada. En forma inversa, un octeto enviado a un periferico es convertido en informacion serie en una linea externa.

Los procesadores centrales actuan en forma simultanea y procesan identica informacion. Deben, en consecuencia, en todo momento coincidir en sus resultados.

Para manejar caracteres Baudot (de 5 bits) y disponer de un repertorio de ordenes y mensajes, se eligieron computadoras de 8 bits de largo de palabra. Estas microcomputadoras, disponibles en unidades de una placa, ofrecen una solucion modular y de bajo costo para un sistema distribuido de esta naturaleza.

Como la informacion se intercambia por octetos, en la forma sincronica y paralelo, para realizar esta funcion, la Central posee relojes de tiempo real (no estan indicados en la figura 1).

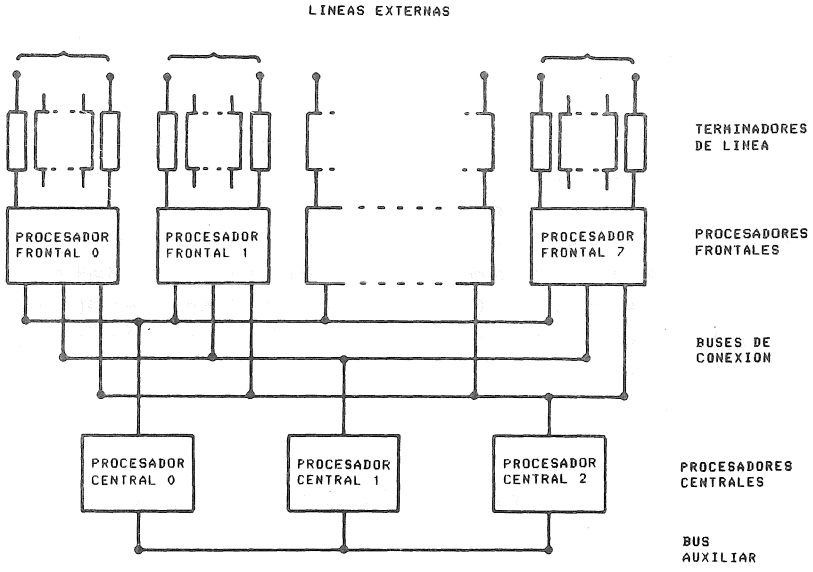


FIGURA 1: Arquitectura del Sistema

Los relojes de tiempo real, que se encuentran triplicados y su accion ocurre por mayoria, tienen las siguientes funciones principales:

a) marcar el comienzo de un ciclo de transferencias.
b) sincronizar la comunicacion entre centrales y perifericos.
c) suministrar la base del sistema de tiempos de las maquinas.

La accion de los relojes de tiempo real se manifiesta a traves de dos mecanismos diferentes:

a) La generacion de una interrupcion a todas las maquinas, centrales o perifericas, cada 1 milisegundo (INTE).
b) La generacion de una bandera (BATRA), cada 138 milisegundos, que señala el comienzo de un nuevo ciclo de transferencias.

La eleccion de estos tiempos se vincula con el manejo de los caracteres Baudot.

La comunicacion entre las centrales y los perifericos

esta sincronizada por la interrupcion de 1 milisegundo. Comienza cuando las centrales reciben la señal BATRA, cada 138 milisegundos. Los perifericos aguardan a que las centrales indiquen que ha comenzado un nuevo ciclo de transferencias.

La comunicacion permite intercambiar octetos que pueden ser caracteres Baudot, ordenes o mensajes.

Como las maquinas centrales deben acceder a diversos perifericos, la comunicacion posee estructura de bus, de tres octetos. Un octeto de direccion permite seleccionar hasta 255 perifericos diferentes, un octeto de datos envia octetos hacia los perifericos seleccionados y un octeto de datos recibe datos de los perifericos. Como existen tres maquinas, el bus se encuentra triplicado. En realidad, cada periferico lee tres datos que provienen de las centrales y envia un dato a los tres buses de las tres centrales.

Las tres maquinas se comunican entre si a traves de un bus auxiliar. Este bus realiza la comunicacion de señales destinados a:

a) sicronizacion de maquinas en el arranque y transferencias.

b) Sincronizacion de los tres relojes redundantes.

III. PROCESADORES FRONTALES (PERIFERICOS)

Los procesadores perifericos realizan las siguientes tareas:

a) Manejan las lineas de la central, es decir, generan y reciben codigos Baudot, generan y reciben pulsos de discado, invierten la corriente de la linea cuando es necesario y miden la distorsion de los caracteres que llegan.

b) Realizan la conversion serie/paralelo y paralelo/serie de los codigos Baudot.

c) Sincronizan los caracteres de manera que la comunicacion con los procesadores centrales pueda realizarse en tiempos fijos.

d) Crean mensajes destinados a las centrales que indican las diferentes etapas de una llamada.

e) Obedecen ordenes de los procesadores centrales que realizan las diferentes etapas de una llamada.

f) Investigan errores internos del propio periferico y generan mensajes de advertencia a las centrales.

g) Investigan la coincidencia de los tres procesadores centrales y generan mensajes de advertencia en caso de error.

Las tareas se organizan en base a dos señales:

1. Señal de interrupcion (INTE). Es una señal de hardware, generada por mayoria de tres señales de interrupcion que se reciben por los buses de comunicaciones. Constituyen la interrupcion que ocurre cada milisegundo.

2. Mensaje de comienzo de transferencias. Este mensaje es una orden enviada por los procesadores centrales. Es decodificada por mayoria por la programacion del procesador frontal. Constituye la indicacion de comienzo de un lapso de 138 milisegundos (BATRA).

El diagrama general de la programacion se muestra en la figura 2. Existen tres areas diferenciadas: una rutina de inicializacion; un programa principal y un programa de interrupcion. El programa principal es un loop que se ocupa, fundamentalmente de la verificacion interna de la memoria RAM y la memoria ROM. Este programa ocupa todo el tiempo que no se emplea en el programa de interrupcion.

Todas las tareas de tiempo real del procesador frontal se organizan como un gran programa de atencion de la interrupcion. En la figura 2 se muestra el diagrama.

La interrupcion comienza con la comunicacion con las centrales y la generacion de mayoria y el analisis de la informacion recibida. Es esta informacion la que organiza la ulterior actividad de la rutina.

El contador de interrupciones, que ordena las tareas, se inicializa con la orden de "comienzo de transferencias" enviada por las centrales. Durante la interrupcion en que se recibe esta orden se envia a las centrales informacion de tipo general (mensaje de no coincidencia de centrales, resultado de chequeo incorrecto de memoria RAM o ROM, chequeo de sus latchs de comunicaciones, distorsion pedida anteriormente de alguna linea). A partir de la recepcion de esta orden, y durante las 16 interrupciones siguientes, se realiza el intercambio de informacion. En la i-esima interrupcion se intercambian caracteres, mensajes y ordenes correspondientes a la i-esima linea de cada proveedor periferico: cada una corresponde a una linea.

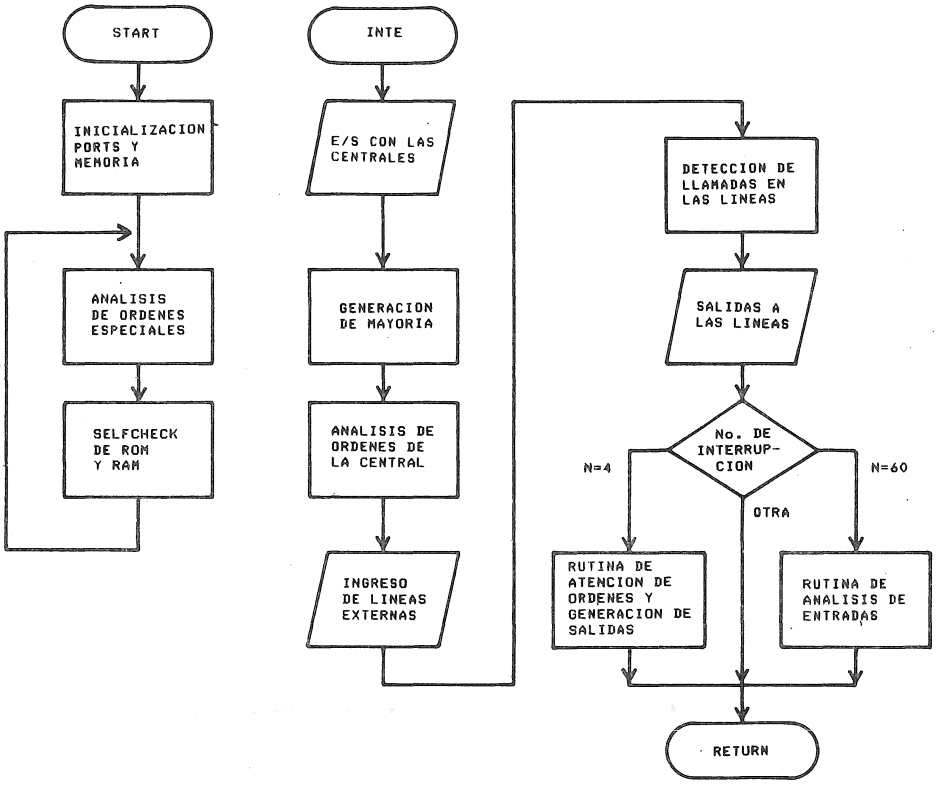


FIGURA 2: Programas del Procesador Frontal

La informacion que llega de las centrales es convertida, por mayoria, en informacion depurada, es generado un diagnostico y es analizado su contenido.

La comunicacion con las lineas externas se realiza en el resto de las rutinas de la interrupcion. Esta comunicacion se realiza a traves de colas.

Los datos muestreados, cada 2 milisegundos, son depositados en una cola de entrada, en memoria. Los datos de salida a las lineas son tomados, cada 5 milisegundos de una cola de salida, en memoria. La cola de entrada deber ser analizada y la de salida generada a intervalos mucho mayores que un milisegundo ya que hacerlo con ese periodo significaria una ineficiencia intolerable. Un periodo razonable es 138 milisegundos que coincide con el elegido para las comunicaciones. Esta eleccion simplifica la programacion, no significa una demora apreciable en la transmision y minimiza el uso de areas para almacenamiento de caracteres.

Las rutinas de analisis de entrada, atencion de ordenes y generacion de salidas ocurren, respectivamente, en las interrupciones numero 60 y 4 del ciclo de transferencias, ver figura 2. Esto hace que las restantes interrupciones, interrumpan estas tareas. Se obtiene asi un gran aprovechamiento del tiempo.

El estado de cada una de las lineas esta definido por una tabla de paramentros. Esta tabla incluye, en particular, la direccion de la rutina de atencion del estado de llamada en curso, tanto en recepcion como en transmision. Un controlador de tiempo real se encarga del despacho de tareas y de actividades de inicializacion de punteros, validez de direcciones de rutinas y verificacion de punteros.

Para el analisis de los datos de entrada, depositados en la cola de entrada, cada rutina de atencion analiza, aproximadamente, 138 milisegundos de datos en cada ejecucion. Define la informaicon recibida y la rutina a ser utilizada en la siguiente ejecucion. La informacion recibida y la direccion de la rutina siguiente se coloca en la tabla de parametros.

En la generacion de datos de salida se procede exactamente igual. En este caso, un puntero define el lugar donde se insertan en la cola de salida los datos generados. Un puntero de salida indica el lugar de donde se tomara la siguiente salida. Durante la generacion de salidas las rutinas correspondientes mantienen una diferencia entre los punteros de carga y salida correspondientes a valores de 40 y 300 milisegundos.

IV. PROCESADORES CENTRALES

Los procesadores centrales realizan las siguientes tareas:

a) Armado y cancelacion de llamadas a traves de los mensajes y las ordenes que intercambian con los perifericos.

b) Tarifado y documentacion de las llamadas completadas.

c) Conmutacion de los caracteres de una comunicacion.

d) Envio de mensajes aclaratorios a las lineas.

e) Suministrar informacion acerca del estado y ocupacion de las lineas.

f) Investigar y documentar errores internos de las

centrales, errores de los perifericos, interrupciones de corriente, temperaturas demasiado altas, etc.

g) Generar documentacion destinada a mantenimiento.

h) Intentar recuperar la marcha de centrales, lineas o perifericos encontrados en falla.

Los requerimientos de tiempo medio entre fallas exigen el uso de unidades centrales redundantes. Se ha implementado una unidad central triplicada por dos razones fundamentales:

1. Una memoria de 8 bits necesita, para recuperar errores, 4 bits adicionales. Esto hace que una logica de diagnostico y recuperacion de errores, para dos unidades centrales, sea comparable en complejidad con una tercera maquina de 8 bits.

2. El reducido costo de una microcomputadora de 8 bits y el caracter modular de tres maquinas iguales frente a una logica de diagnostico y recuperacion de errores.

Si el largo de palabra fuera de 16 bits en lugar de 8, el peso de estos argumentos es menor y de alli que practicamente la mayoria de las Centrales Telex se hayan construido con maquinas de 16 bits y unidades centrales duplicadas.

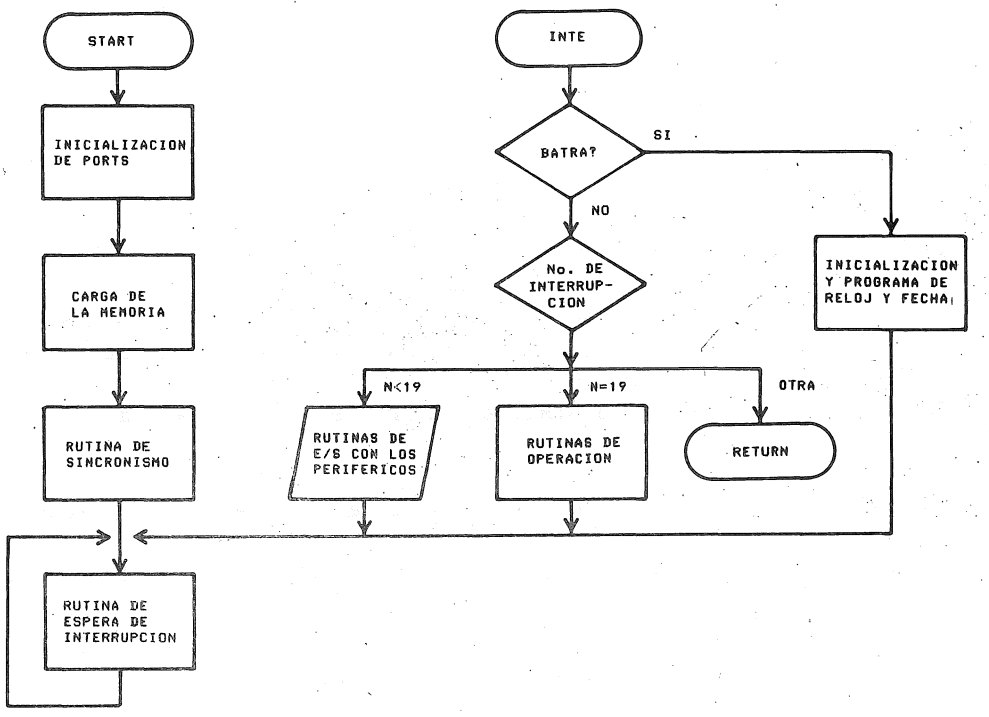


FIGURA 3: Programas del Procesador Central

Si el numero de perifericos es reducido, la confiabilidad depende de los procesadores centrales. A medida que aumenta el numero de perifericos aumenta la probabilidad de que una puerta de comunicaciones quede habilitada en forma permanente, por fallas sobre un bus, provocando su inutilizacion. El numero de perifericos se vuelve entonces el factor dominante en la confiabilidad del sistema. El punto de quiebre depende de la confiabilidad de las comunicaciones y de los computadores centrales.

El funcionamiento de las tres maquinas es sincronico, a menos de la pequeña deriva entre dos instantes sucesivos de sincronizacion de los tres relojes que suministran las interrupciones y banderas de transferencia.

La programacion de las maquinas centrales se muestra en la figura 3.

El programa de inicializacion comienza en START. Luego de inicializar las puertas de entrada/salida se carga la memoria. A continuacion se ejecuta el programa de sincronismo, destinado a lograr que las tres maquinas centrales arranquen sincronizadas. Los detalles de este proceso se analizan en la seccion VI. Este Sincronismo en el arranque se logra a menos de ± 1 microsegundo.

La operacion continua sincronica porque la señal BATRA (cada 138 milisegundos) restaura el sincronismo. Todas las tareas realizadas por las unidades centrales tienen una duracion inferior a 138 milisegundos.

Antes de finalizar el tiempo disponible, cada unidad se coloca en estado HALT, a esperar una señal BATRA para comenzar un nuevo ciclo en sincronismo.

Los cristales de las computadoras tienen una precision mayor que 0,1%. La tarea mas larga en cada ciclo es menor que 138 milisegundos. El maximo desfasaje entre dos maquinas al final de una tarea es menor que 138x0,001=138 microsegundos. No se ha tomado ninguna precaucion para el desfasaje entre tareas aun cuando esto puede disminuir las incidencias de posibles fallas correlacionadas.

La organizacion de las tareas de las maquinas centrales se realiza en el programa de interrupcion.

El programa de la interrupcion esta regulado por contador y por la aparicion de BATRA. Con la aparicion de BATRA se coloca el contador en cero, se inicializan punteros y se ejecuta el programa de reloj y de fecha.

Las interrupciones 1 hasta 18 ejecutan el programa de entrada salida. En la interrupcion 19 se lanza el resto de la operacion de la central. Toda interrupcion ulterior, hasta una nueva aparicion de BATRA, no realiza otra actividad que incrementar el contador y regresar.

En el resto del tiempo se procede a realizar toda la actividad de atencion de llamadas, atencion de resets de centrales y perifericos, atencion de consolas y, finalmente, tareas de enrutamiento, tarifacion, documentacion de llamadas y documentacion en general.

Para administrar el tiempo de operacion se emplean dos tecnicas diferentes. Cada linea necesita atencion en un ciclo de 138 milisegundos, de otro modo se perderian caracteres. Como el limite superior de terminales previsto de 1024, para lograr la atencion de todas las lineas en un ciclo, se limito el tiempo maximo de ejecucion de cada rutina a 100 microsegundos. El estado de cada linea ha sido, en muchos casos, subdividido en varios subestados a fin de lograr estos tiempos de ejecucion.

Para las tareas de enrutamiento o documentacion de llamadas que son de tiempo de ejecucion mucho mayor (del orden de varios milisegundos) se procede en forma diferente. Se dispone de un algoritmo para la determinacion del maximo tiempo de ejecucion para una tarea. Las maquinas realizan una estimacion de este tiempo y efectuan un numero limitado de estas tareas.

V. BUSES DE COMUNICACION

La Central Telex posee dos tipos de buses de comunicacion: comunicacion con los procesadores frontales e intercomunicacion entre procesadores centrales.

En todos los casos, es necesario asegurar que ocurran los siguientes hechos:

a. Una falla en un dispositivo no debe anular las comunicaciones del conjunto.

b. Un dispositivo apagado o desconectado no debe anular las comunicaciones ni generar señales que no se puedan interpretar.

Cuando se desconecta un equipo se tiene el mismo estado logico que un nivel bajo en la entrada. De esta manera, con un adecuado diseño de la logica y la programacion, se puede tolerar la desconexion de cualquiera de los buses de comunicacion triplicados de la maquina. Estos criterios han sido adoptados en el diseño.

VI. RELOJES Y SINCRONIZACION DE LAS TRES MAQUINAS

El reloj esta triplicado (un reloj por central) y utiliza un metodo conocido de generacion de señales sincronicas [2] que se ilustra en la figura 4. Las señales que se generan son:

1. Señal de 1 milisegundo de periodo que se utiliza como interrupcion para todas las maquinas.

2. Señal de 138 milisegundos de periodo (BATRA) que marca el comienzo de las transferencias de datos, ordenes y mensajes entre centrales y perifericos.

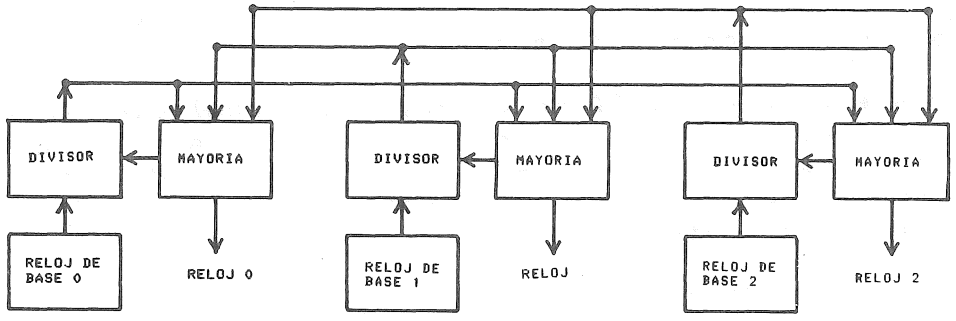


FIGURA 4: Relojes Triplicados

El reloj primario para la generacion de la interrupcion es el reloj de la microcomputadora central. Un contador divide esta señal. 1024 cuentas equivalen a 1 milisegundo. Al llegar a la cuenta 1023 el contador genera un pulso y deja de contar. El contador es reseteado por una señal que es la mayoria de las tres señales de los tres relojes. Esta configuracion permite el funcionamiento con tres relojes funcionando correctamente o con dos relojes funcionando correctamente, independientemente de la falla del tercero. Permite, ademas, sincronizar las tres señales de interrupcion a menos de un ciclo del reloj de base.

El metodo utilizado para la generacion de BATRA es similar al usado en la generacion de la interrupcion. El reloj primario es la señal de interrupcion. El contador divide por 138 y emplea tambien el esquema de la figura 4.

Para la sincronizacion de las maquinas usan la señal de tiempo BATRA. El mecanismo de sincronizacion es el siguiente:

a) La sincronizacion comienza con un loop de espera de la señal BATRA.

b) Una vez encontrada se espera mas de 1 milisegundo.

c) Luego de la espera se envia a las otras dos maquinas la señal de sincronismo y se comienza la espera de la siguiente señal BATRA.

d) Recibida la segunda BATRA se leen las señales de sincronismo provenientes de las otras dos maquinas. Si ambas señales estan presentes, la maquina comienza su funcionamiento normal. Si solo esta una de las señales, comienza la espera de la otra señal de sincronismo, durante un maximo de 10 BATRAS.

e) Transcurridas 10 BATRAS se supone que la tercera maquina no esta en condiciones de marcha y las dos presentes comienzan su funcionamiento normal.

La espera de 10 BATRA es mayor que cualquier diferencia de tiempo en el arranque de las maquinas. La espera (en b) para generar la señal de sincronismo evita problemas de azar en la lectura por parte de las otras maquinas, que leen las banderas a continuacion de detectar BATRA.

Es conocido el caracter nocivo, para los sistemas redundantes, de las fallas de hardware que generan una señal cierta para una unidad y una falsa para otra. Este tipo de fallas es siempre posible, pero se ha cuidado que no ocurran en dos situaciones importantes: cuando se desconecta una unidad y cuando se inicializa el sistema.

Cuando se quita la alimentacion a una maquina o cuando se desconecta el bus que encamina las señales de sincronismo, las otras dos maquinas deben ver el mismo nivel logico en las entradas correspondientes a la tercera maquina. Este nivel debe ser correspondiente a ausencia de sincronismo. Todas estas funciones se logran con un separador adecuado para las señales.

Al inicializar el sistema, los ports de salida (que son programables) estan en un estado de alta impedancia, por lo tanto la salida se interpreta como circuito abierto (1 logico en los separadores). Cuando se configura la port, automaticamente pasa a cero. Se desea que la salida no indique sincronismo cuando se desconecta la fuente de poder (es decir, en circuito abierto).
Por lo tanto, en el momento del arranque siempre existe un pulso para que este pulso no genere nunca un sincronismo falso, se espera una BATRA y luego de una espera conveniente se inicializan las ports y la señal de sincronismo. Esto asegura que el pulso no valido no ocurra en la zona en que cada maquina analiza los bits de sincronismo.

VII. DETECCION DE ERRORES Y REINICIALIZACION

La Central Telex posee mecanismos de deteccion de errores de los procesadores y de intentos de retoma de la marcha. Estas acciones ocurren por un proceso interactivo entre procesadores centrales y perifericos.

Los tres procesadores centrales verifican el comportamiento de los procesadores frontales usando tres criterios. El primero consiste en investigar las comunicaciones. A estos fines el procesador frontal envia, alternativamente, los bytes 0101 0101 y 1010 1010 en momentos oportunos. Si las centrales no detectan un mensaje valido acumulan un error al procesador frontal correspondiente. De esta menera se tiene una razonable vigilancia de las comunicaciones.

El segundo criterio de error consiste en investigar la secuencia de bytes recibidos. Todo error de consistencia de la secuencia aumenta el contador frontal.

El tercer criterio de error proviene de autodiagnostico del procesador frontal. El mensaje de errores internos de la memoria (ver figura 2) aumenta el contador correspondiente.

Cuando la cuenta de errores de un procesador frontal supera 16, se inicia una accion de reinicializacion. Esta accion consiste en enviar una orden de RESET y verifica que el procesador responde con el codigo de RESET REALIZADO. En caso de una falla persistente de hardware, no se lograra recuperar la marcha del procesador, pero se dispondra de la informacion que permita diagnosticar el origen de la falla.

Los procesadores centrales son vigilados por el conjunto de los procesadores perifericos. Cada procesador frontal actua como un votador para la informacion recibida desde las tres centrales: la reciben por triplicado y la procesan por mayoria. En caso de no coincidencia se envia un mensaje a las tres centrales. Cada central los acumula al contador de errores correspondiente.

Una central solo es reinicializada cuando existe un acuerdo de las otras. De esta manera se evita que una central en falla pueda perturbar otra en funcionamiento correcto. Para lograr esto, cuando una central j encuentra que la central i ha acumulado 256 errores, envia una señal de reset de hardware. Cada central posee una linea de reset de las otras dos. El AND de estas dos señales esta conectado al reset verdadero de la maquina.

Una vez ocurrido el reset, es transferido el contenido de la memoria de las otras dos maquinas a traves de un

mecanismo de comunicacion, paralelo, sincronico, muy eficiente. Una central se resetea, a lo sumo, tres veces. Solo la intervencion del operador permite cancelar este limite. Esta cota permite evitar que dos centrales no resulten sobrecargadas en su tarea de resets a una tercera central, fuera de servicio, por falla persistente.

Puede pensarse que un punto debil del mecanismo de vigilancia se encuentra en un caso de falla de un procesador periferico que de por resultado un mensaje "central mal". Resulta muy improbable que una falla en un procesador frontal genere este mensaje para una central que funciona correctamente y, a su vez, no genere otros errores que lo coloquen muy rapidamente fuera de servicio. Por lo tanto, todo mensaje de error cursado a una central, se considera valido. Son los limites de 16 y 256 errores quienes resuelven un posible conflicto de esta naturaleza.

VIII. CONCLUSIONES

El diseño e implementacion de sistemas redundantes, con deteccion y recuperacion de fallas es, sin duda, el camino que abre mas perspectivas para el tratamiento de datos en los proximos años.

La experiencia practica obtenida en este proyecto de Central Telex de 128 lineas permite encarar en el momento actual, el diseño de un equipo de 512 lineas. Se extendera a este caso los conceptos de triplicacion, recuperacion de errores y distribucion de tareas. Se proyecta, ademas, mejorar la tolerancia a errores de programacion de todo el sistema, tecnica ya empleada en la configuracion actual.

REFERENCIAS

[1] Tecnica Telegrafica, Libro Naranja, CCITT, 1978.

[2] Synchronization and Matching in Redundant Systems.
D. Davies and J.F.Wakerly.
IEEE, Transactions on Computers, Vol C-27, June 1978.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

LA ARQUITECTURA DE UN COMPUTADOR CONCURRENTE

Sergio Mujica
Javier Pinto
Departamento de Ciencia de la Computación
Instituto de Matemática
Pontificia Universidad Católica de Chile

INTRODUCCION

Un programa concurrente hace que se distribuya actividad en varios procesadores, es decir, dado un programa concurrente este será ejecutado por muchos procesadores, los que trabajarán en forma paralela, (también se habla de computación paralela) ejecutando simultáneamente diferentes computaciones, lo que obviamente reduce la complejidad de tiempo de los algoritmos a algo mucho menor (órdenes de magnitud) que en los algoritmos secuenciales basados en el modelo de von Neumman.

El propósito de este trabajo es dar una descripción de la arquitectura de un computador que soporte un lenguaje para escribir programas concurrentes.

REPRESENTACION DE LAS COMPUTACIONES

Normalmente se representan las computaciones como una secuencia de acciones que cambia el estado de un conjunto de objetos. También es posible representarlas individualizando el cambio de estado de los objetos, esto es, podemos pensar que se realiza una computación alterando o cambiando el estado de un número finito, aunque no necesariamente acotado, de objetos que representan el ambiente en que se realizan nuestras computaciones, estos cambios de estado puede significar acciones predeterminadas (E/S), o bien

---

Este trabajo ha sido financiado por la Dirección de Investigaciones de la Pontificia Universidad Católica de Chile (DIUC), contrato N82/79

resultar de la acción misma del objeto o de otro objeto.

OBJETOS DE DATOS

Para seguir adelante daremos una definición más precisa de lo que entenderemos por un objeto de datos.

- Tipo de Datos
  Un tipo de dato primitivo es un conjunto de datos primitivos definido por las operaciones que se pueden usar sobre ellos; a diferencia del concepto tradicional estas operaciones sólo pueden alterar el estado del objeto al que corresponde el tipo de datos. Los tipos de datos primitivos son aquellos propios de la máquina, es decir, son inherentes a los elementos de proceso, (i.e., la máquina es capaz de manejarlos).

  Un tipo de dato derivado debe representarse a nivel de programa mediante una estructura de objetos de datos cuyos tipos pueden ser primitivos o derivados, esto se verá con más claridad una vez que se defina objeto de datos.

- Objeto de Datos
  Un objeto de datos tiene asociado un tipo de dato que dice cuales son las operaciones que pueden modificar su estado. Además estos objetos poseen un comportamiento.

  Esquemáticamente (fig. 1), podemos representar un objeto de datos mediante un contorno rectangular el cual tiene buzones (sockets) de entrada y buzones de salida; el objeto de datos será activado en el momento en que se haya depositado información en alguno de los buzones de entrada, lo que motivará la ejecución de una acción por el objeto de dato y que generará resultados que el objeto depositará en los buzones de salida.

  Llamaremos token al ente que transporta información, que es posible depositar en un buzón, además cada buzón tendrá asociado un tipo de dato que indicará el tipo de información que por él puede circular o que él puede manejar.

INTERCONEXION DE OBJETOS

Para que exista un flujo de información (flujo de datos) se hará necesario interconectar objetos, esta interconexión se hará entre objetos (que no necesariamente estará asociado al mismo tipo de datos) mediante enlaces que asocian un buzón de salida de un objeto, y un buzón de entrada de otros objetos, donde obviamente los buzones así asociados deberán corresponder al mismo tipo de datos para que haya un correcto flujo de datos.

ESTRUCTURA DE UN PROGRAMA

Un programa es una estructura de objetos de datos. Para definir una estructura de este tipo tenemos que seguir los siguientes pasos:

1. Definir las clases de objetos que van a formar parte de nuestra estructura.

2. Definir la estructura: como se relacionan los objetos entre sí; estas relaciones corresponden a enlaces de comunicación.

3. Si al definir el programa hemos usado una clase de objetos cuyo tipo asociado es derivado, tenemos que especificar una representación del tipo de datos.

Ejemplo:
Supongamos que tenemos un archivo de datos y deseamos contar el número de caracteres que éste contiene.
Podemos identificar claramente tres objetos de datos, estos son:

- Archivo de entrada: donde se encuentra el texto a leer
- Contador : cuenta el número de caracteres
- Archivo de salida : donde se imprimirán los resultados.

En base al distinto tipo de información que requieren los objetos para actuar, podemos definir los enlaces de comunicación de los tres tipos de datos antes especificados, además podemos identificar en cada uno de los objetos los buzones correspondientes (fig. 2).

- Archivo de entrada: no requiere de un buzón de entrada, es decir, bastará crear este objeto para que comience a actuar, tendrá un buzón de salida, en el cual el objeto depositará los tokens que lleva la información (caracteres de entrada) contenida en el archivo de entrada.

- Contador: tendrá un buzón, el cual tendrá asociado el tipo de caracter (char), y será donde se depositarán los tokens con los caracteres que el contador contará; el contador recibe un caracter especial (Z) que le indica que debe terminar de contar, entonces deposita el token con el resultado de su acción en el buzón de salida.

- Archivo de Salida: tendrá un buzón de entrada, por el que ingresa un entero, que éste imprime.

Esquemáticamente (fig. 2) podemos ver la forma de interconexión de los objetos ya mencionados; que forman la estructura del programa que deseamos construir.

ESPECIFICACION DE OBJETOS

Para que un objeto quede completamente especificado, es necesario, definir en términos del lenguaje (para los ejemplos usaremos sintaxis ad hoc) el estado inicial del objeto, los buzones (sockets), el tipo de dato a que está asociado el objeto y además debemos describir su comportamiento.

En el ejemplo los objetos se puededen especificar de la siguiente manera:

```
object infile : input
init infile ← open ('file name')
sockets out (c : char)
(eof)    c← ' Z'
(¬ eof) c ← read
end
```

```
object contador : integer
init contador ← 0
sockets in (c : char), out (K : integer)
(c = ^Z) K ← contador
(c ≠ ^Z) new contador ← contador + 1
end
```

```
object outfile : output
init outfile ← open ('file name')
sockets in (K : integer)
( ) print (K)
end
```

DESCRIPCION FORMAL DE UN OBJETO

1. Identificación del objeto

   Se da el nombre del objeto y se especifica el tipo de dato a que éste está asociado, en el ejemplo:

   ```
   object infile : input;
   ```

   donde input corresponde a un tipo de dato primitivo

2. Estado inicial

Se especifica el estado inicial del objeto, en el ejemplo:

init infile ← open ('file name')

Especifica que el estado inicial del objeto es open, siendo open una operación que corresponde al tipo de dato input, que requiere un parámetro (nombre del archivo).

3. Especificación del contorno

Se especifican los buzones (sockets) que el objeto dado posee, indicando si ellos corresponden a buzones de entrada o de salida e indicando el tipo de dato asociado a cada buzón; en el ejemplo:

sockets out (c : char)

4. Especificación del comportamiento

El comportamiento de un objeto es un conjunto de pares ordenados B : <p, A> donde p es un predicado (condición) y A es un conjunto de acciones. Una acción posible es depositar el token en un buzón.

En el ejemplo el conjunto es:

$$B = \{ \langle eof,\ c \leftarrow '\hat{}Z' \rangle\ ,\ \langle M\ eof,\ c \leftarrow read \rangle \}$$

donde los predicados son p_1 = eof y p_2 = eof y las acciones correspondientes son:

$$a_1 = c \leftarrow '\hat{}Z';\ a_2 = c \leftarrow read$$

Al crearse una instancia de un objeto se ejecuta lo especificado en init, es decir, pasa al estado inicial.

Una instancia de objeto entra en actividad al recibir un token en uno de sus buzones de entrada. Entonces el objeto evalúa los predicados del conjunto B. La evaluación de un predicado es exitosa si él es verdadero y se dispone de todos los elementos necesarios para evaluarlo; falla si no hay tokens suficientes para evaluarlo o es falso.

La máquina selecciona no determinísticamente un par tal que su predicado se haya evaluado exitosamente y ejecuta las acciones de A.

Además cada objeto tiene un buzón de salida asociado con un buzón de entrada a sí mismo (fig. 3), el que es utilizado para cambiar el estado del objeto, este cambio se realiza especificando, como en el ejemplo anterior:

new object name ← expression

Que equivale a poner un token, con el valor resultante de evaluar la expresión, en el buzón new X cambiando así el estado del objeto X (ver especificación del objeto "contador" en el ejemplo anterior.

DEFINICION DE ESTRUCTURAS

Una estructura es un grafo dirigido, que puede variar a medida que la computación progresa.

Así, una estructura S estará compuesta por arcos y nodos. Los arcos son enlaces de comunicación entre objetos a través de los cuales los tokens transportan datos de un objeto a otro. Los arcos conectan un buzón de salida en un objeto con un buzón de entrada en otro objeto.

En la estructura existen dos tipos de nodos: objetos y reproductores.

Los objetos corresponden a la descripción anterior y los reproductores son nodos a través de los cuales se puede hacer variar la estructura.

Formalmente los podemos definir como sigue:

Def. Un reproductor es un par ordenado [O, S] en que:
S es una estructura en que uno de los objetos tiene un buzón distinguido B y O es un objeto tal que:

i) su procedimiento de inicialización pone su estado en valor nulo.

ii) tiene sólo un buzón de entrada denominado in.

iii) su comportamiento es el siguiente:

(se recibió el token)

- se crea instancias de los objetos y reproductores que forman parte de la estructura S.
- se instalan los enlaces definidos en S
- se instala el enlace que llega al buzón del reproductor en el buzón distinguido B.
- se deposita el token recibido por el reproductor en el buzón B.

Entonces, podemos decir que la definición de una estructura consta de las siguientes partes:

a) Definición de las instancias de objeto que forman parte de la estructura, identificándolas adecuadamente.

b) definición de los reproductores

c) definición de enlaces.

Por ejemplo, podemos definir la estructura del programa que utilizáramos anteriormente, como sigue

```
Structure
Objects
 ⟨i : infile, cont : contador, o : outfile⟩
links
⟨i.c to cont.C⟩
⟨cont.K to 0.K⟩
end
```

Como un segundo ejemplo podemos considerar un arbol binario. Para que la estructura crezca, un reproductor debe generar un nodo con dos hijos reproductores, como esquematiza la figura 4.

Esta estructura se puede formalizar como sigue:

```
Structure
gen⟨F (objects  n : node
gens  ⟨e, r : self⟩
links ⟨F.in to n.root⟩
      ⟨n.ls to l.in⟩
      ⟨n.rs to r.in⟩
 )⟩
links   ⟨ ... to F.in⟩
end
```

REPRESENTACION DE TIPOS DE DATOS

Como se dijo anteriormente un tipo de datos está definido por un conjunto de operaciones que pueden tener parámetros.

Representaremos un tipo de datos como una estructura en la que existirá un contorno externo que contendrá buzones para los parámetros de las operaciones.

Al definir la estructura, se deben especificar, enlaces entre los buzones de parámetros y buzones de entrada en la

estructura (entry points).

Por ejemplo, una representación para objetos del tipo bynary-search-tree, con la única operación insert;

```
type bynary-search-tree
  op insert (n : integer);
object node : integer
sockets in (root : integer)
out        (ls, rs : integer);
init new node ← root
(root > node) rs ← root
(root < node) ls ← root

Structure

    gens <F (objects  <n : node>
             gens     <l, r : self>
             links    <F. in to n.root>
                      <n.ls to l.in>
                      <n.rs to s.in>
             )>
    links
   <insert.n to F.in>
    end structure
end type
```

ELEMENTOS DE PROCESO

La arquitectura que se propone consiste básicamente en una red de elementos de proceso, cada uno de los cuales puede representar a un objeto.

Un elemento de proceso (EP) está formado por tres componentes (figura 5).

E/S representa los buzones, P es capaz de realizar las acciones del comportamiento y M es una memoria.

Denominaremos n-EP a un EP tal que su unidad E/S tenga n líneas de comunicación (una línea de comunicación tiene en

general una cantidad de líneas paralelas).

Cada p_i en la unidad E/S se asocia con una tupla $\langle d, t, n, Q, S \rangle$ en que:

.d es entrada o salida

.t es el tipo de datos del buzón

.n es el nombre del buzón

.Q a) si d = entrada

entonces Q es en cada instante la secuencia de tokens que existe en el buzón.

b) si d = salida

entonces Q es la secuencia vacía.

.S a) si d = entrada

entonces S es la secuencia vacía

b) si d = salida

entonces S es una secuencia de nombres de EP que indica el camino que un token depositado en el buzón debe seguir para llegar a su destino.

Este concepto se desarrollará más adelante al definir la representación y creación de enlaces

Aunque el objeto no necesita conocer los nombres de sus vecinos, se incluye S por eficiencia.

Los datos necesarios para representar un objeto se definen usando una sintaxis similar a PASCAL como sigue:

```
type objeto
     record
       t : tipo;
       b : Array of buzón;
       I : inicialización
       c : Array of comportamiento
end
type buzón = Record n . nombre; ti : tipo end
type comportamiento =
     record
       C : condición;
       A : set of acción
end
```

LA ESTRUCTURA DE LOS RETICULADOS

Los elementos de proceso son los nodos de un reticulado (grafo no dirigido en que de cada nodo sale el mismo número de arcos). Un arco que conecta dos nodos representa una línea de comunicación entre ambos elementos de proceso.

Clasificaremos los reticulados según el número de arcos que salen de cada nodo, de acuerdo a la siguiente definición.

Def. Un n-reticulado es aquel en que cada nodo tiene n arcos de salida, y cada par de nodos tiene a lo más una conexión.

La figura 6 ilustra ejemplos de n-reticulado para n = 4 y 6.

Consideraremos que los elementos de proceso de un n-reticulado se agrupan en unidades que llamaremos n-células.

Def. Una n-célula es una agrupación de n + 1 elementos de proceso en que:

a) Se distingue un elemento central que tiene todos sus arcos conectados a otro nodo de la n-célula.

b) Cada elemento distintio del central está conectado al nodo central y a dos vecinos de la manera que ilustra la figura 7.

c) $n \geq 4$

Construiremos reticulados interconectando un conjunto de n-células.

Físicamente, un reticulado de elementos de proceso será un único chip (pastilla).

CREACION DE OBJETOS

Se asigna un objeto a un elemento de proceso, cargando la representación correspondiente al objeto en el elemento de proceso. Cada instancia de objeto asociada a un elemento de proceso tiene un nombre único, el que corresponde exactamente al nombre del elemento de proceso (su "dirección")

REPRESENTACION DE ENLACES

Un enlace entre dos objetos se representa como una secuencia de nombres de objeto que conectan los dos EP involucrados en el enlace dentro del reticulado

Por ejemplo, un enlace entre los objetos A y B en la figura 8 se puede representar como la secuencia ⟨6,10,11,15⟩ otra sería ⟨6,7,11,15⟩ o ⟨6,7,8,12,16,15⟩

Debido al método de creación de enlaces que se enuncia en el próximo punto, siempre los EP que se nombran en la representación de un enlace tendrán la instancia de un objeto asignada a ellos.

CREACION DE ESTRUCTURAS

Un reproductor, ejecutándose en un EP es capaz de crear una estructura. La creación de una estructura involucra crear instancias de objetos de enlaces.

Para crear una instancia de un objeto, el reproductor envía la representación a un EP vecino al suyo. El EP destino se selecciona de acuerdo a:

a) si algún EP vecino está desocupado (i. e., no hay una instancia de objeto en él), se selecciona.

b) si todos los vecinos están ocupados, se selecciona uno al azar y se le envía el objeto. Este lo redirige de acuerdo a estas mismas reglas.

Durante el viaje de la representación del objeto hasta un EP desocupado, se registra el camino seguido. Una vez instalada la instancia de objeto, se envía de vuelta por el mismo camino la localización (nombre) del EP en que se instaló la instancia.

Para establecer un enlace, el reproductor despacha una sonda, la que viaja en el reticulado buscando los objetos que se deben interconectar. Una vez que ubicó uno de ellos comienza a registrar el camino seguido de tal manera que al encontrar el segundo de ellos, ya conoce un camino que los une.

SOBRE LOS ELEMENTOS DE PROCESO

Es necesario, admitir una posible heterogeneidad de los EP en relación con las operaciones que cada uno es capaz de realizar. Esto se debe a que si todos fueran homogéneos, tendrían que ser capaces de realizar todas las operaciones posibles y serían, cada uno de ellos, demasiado complicados, lo que implicaría que el número de EP por chip sería pequeño.

Podemos especializar los EP de acuerdo a los tipos de dato que sean capaces de manejar. Esta política tiene la ventaja de reducir considerablemente la complejidad de un elemento de proceso.

ELEMENTOS FRONTERA

Es evidente que el número de EP que se puede poner en un chip está limitado. Para interconectar chips para formar un reticulado mayor se ponen en los extremos de un reticulado pequeño elementos frontera. La figura 9 ilustra un 4-reticulado de 9 elementos de proceso, con sus nodos de frontera.

Un elemento frontera puede tener uno ó dos vecinos:

a) si tiene un vecino, actúa como un muro, haciendo rebotar los mensajes que le envía su único vecino.

b) si tiene dos vecinos, uno de ellos debe ser un elemento frontera en un chip distinto. La comunicación entre elementos frontera en chips distintos es obviamente mucho más lenta que con otro del mismo chip.

Los elementos frontera permiten la comunicación entre elementos de proceso ubicados en chips diferentes.

CONCLUSIONES

La tecnologia VLSI actual [Mead-Conway 80], hace posible la realizacion fisica de una arquitectura como la propuesta.

Por otro lado el lenguaje que hemos descrito y que es aquel definido en [Mujica 80], nos permite escribir programas altamente concurrentes utilizando el enfoque de objetos, cuya deseabilidad es evidente [Wegner 79].

REFERENCIAS

[Mead-Conway 80] Carver Mead. Lynn Conway
"Introduction to VLSI systems"
Addison-Wesley, Reading, Massachussets, 1980.

[Mujica 80] Sergio Mujica, "Un modelo semántico para computación concurrente", Actas de la séptima conferencia latino-americana de informática, Caracas-Venezuela, Febrero 1980.

[Wegner 79] Peter Wegner, "Programming Languages-Concepts and research Directions", in Research Directions in Software Technology, (p. Wegner, ed.), The MIT Press, Cambridge, Mass, 1979.

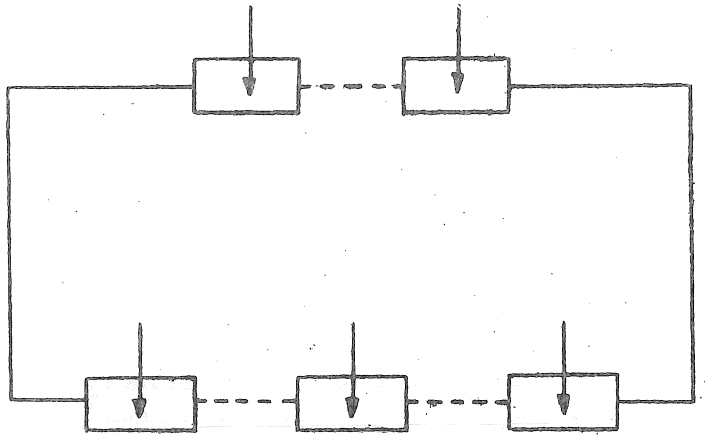


FIG. 1

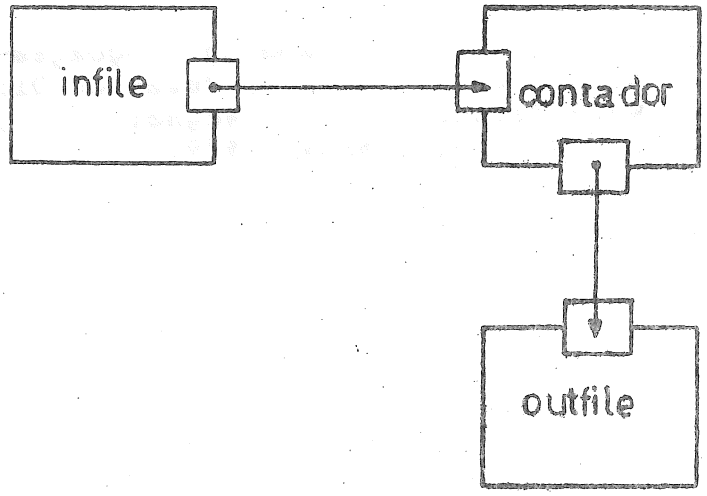


FIG. 2

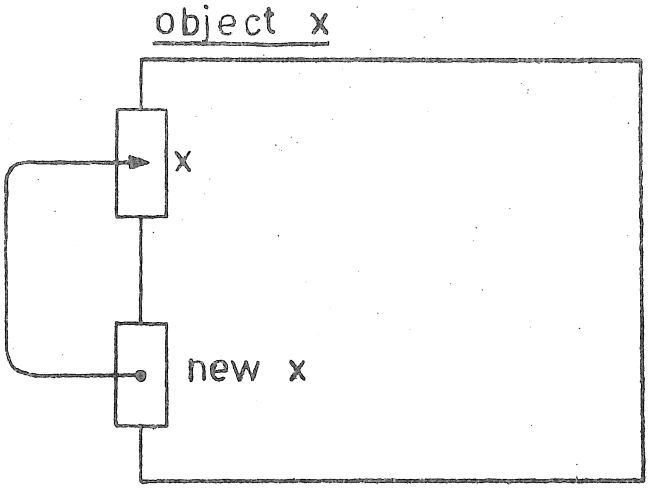


FIG. 3

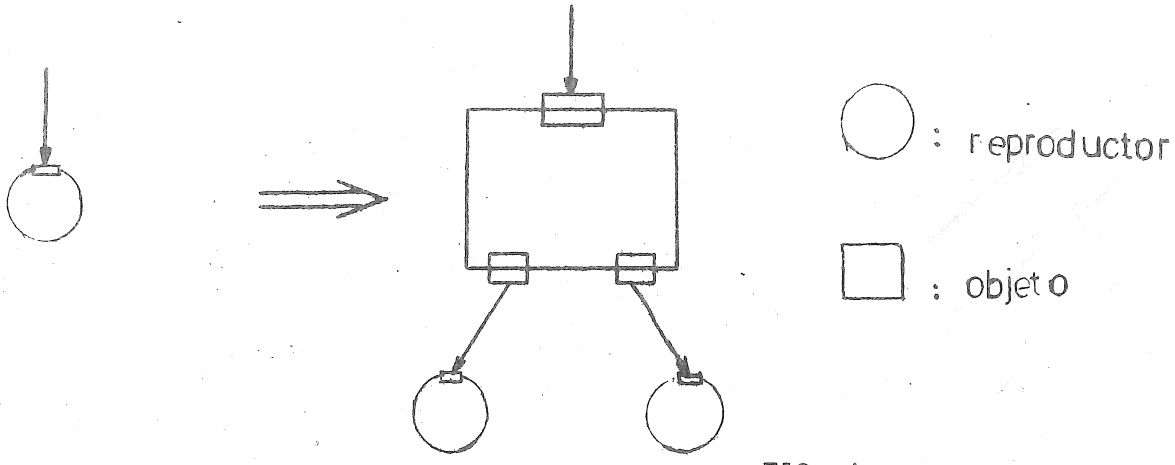


FIG. 4

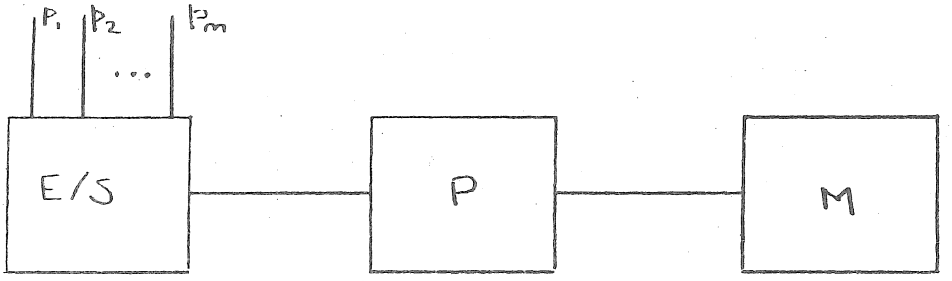


FIG. 5

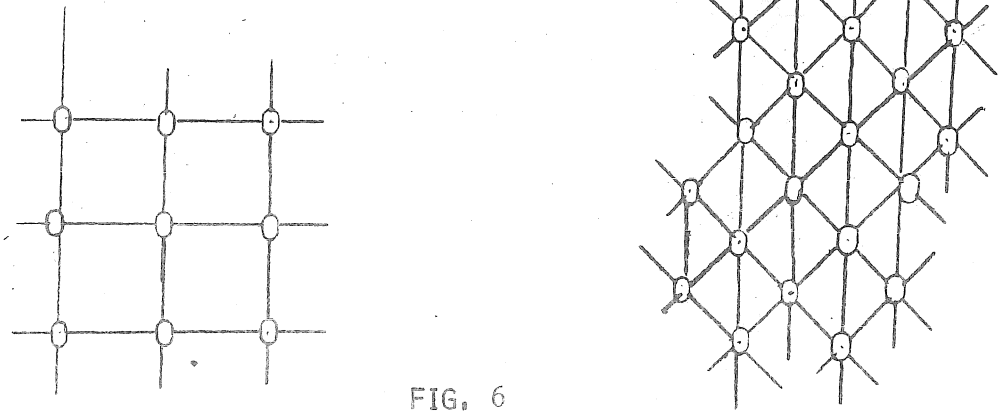


FIG. 6

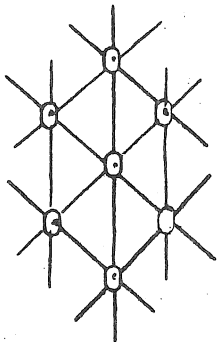
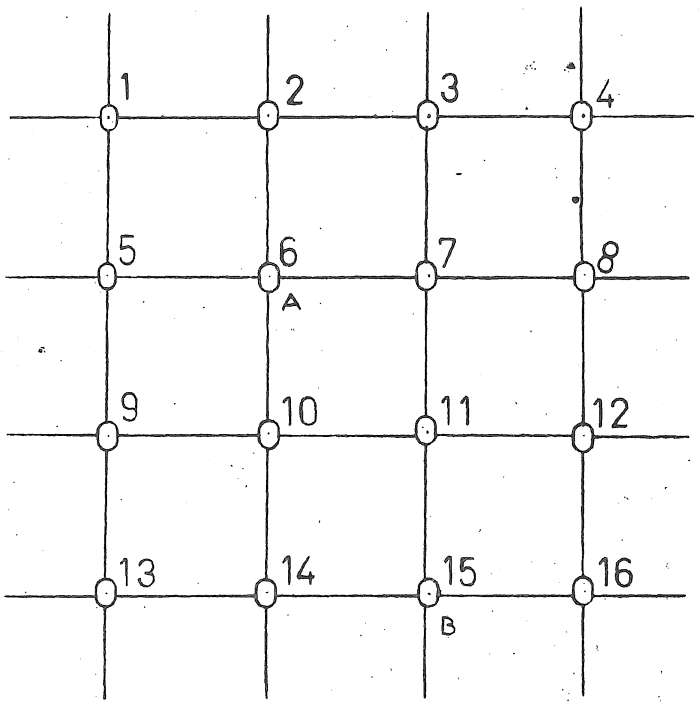


FIG. 7



Un reticulado FIG. 8

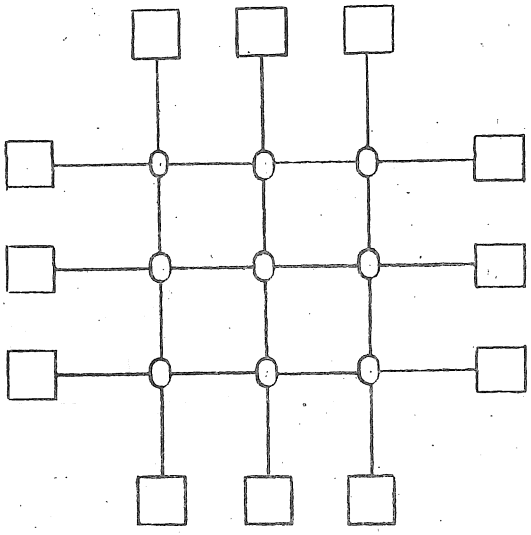


FIG. 9

O : Elemento de proceso

□ : Elemento frontera

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

DISEÑO DE LA INTERFASE PARA LA RED LOCAL UNIANDES

Leonardo Araújo
Jairo Fernández
Roberto Pardo

Departamento de Ingeniería de Sistemas
Universidad de Los Andes
Apartado Aéreo 4976, Bogotá - Colombia.

OBJETIVO

La tecnología de hardware (1) y software para interconexión de computadores (redes) ha avanzado a pasos gigantezcos en la última década. Muchos adelantos aplican a redes de amplia cobertura geográfica, donde los nodos son computadores relativamente grandes y costosos, donde los nodos están dispersos por ejemplo en un país, y donde las decisiones de diseño de hardware y software de comunicación tienden a minimizar un costo inherentemente alto de comunicación.

En los últimos 5 años, con la proliferación de máquinas medianas y pequeñas, se ha despertado un gran interés por la interconexión de computadores localizados en áreas pequeñas (un edificio, un sector de la ciudad). En este tipo de redes locales las decisiones de diseño e implantación tanto del hardware como del software de comunicación son un tanto diferentes. Por ejemplo, en vez de usar conmutadores y/o procesadores de comunicación los cuales son costosos por cuanto el mismo sistema y la comunicación es costosa, se usan Interfases sencillas y poco costosas.

En este artículo se presentan las principales decisiones de diseño de una Interfase que se usará para la interconexión local de varios computadores de la Universidad de Los Andes, Bogotá. Adicionalmente se explicará por qué es interesante realizar este tipo de proyectos en países como los nuestros.

TERMINOS CLAVES : Interfase, Sistemas Distribuídos, Protocolo, Red Local.

---

(1) Se usarán algunos términos del inglés como "hardware", "software", "buffer" etc., cuando se considere que no tienen una traducción ampliamente aceptada.

1. INTRODUCCION

Este artículo describe el diseño de la Interfase que une los diferentes nodos que conforman la "Red local Uniandes". Una red local, como su nombre lo indica, es una red de comunicación de datos, que cubre un área geográfica pequeña (un edificio o un sector de la ciudad). Este tipo de redes generalmente provee un ancho de banda alto sobre un medio de transmisión poco costoso. (En (CLAR78) se habla en más detalle sobre redes locales).

La red local Uniandes es una red experimental, actualmente en fase de diseño, que nació como un esfuerzo conjunto entre los Departamentos de Ingeniería Eléctrica y de Sistemas de la Universidad de Los Andes. Los objetivos principales del proyecto son :

1. Experimentar en el diseño e implantación de tecnología de hardware y software de redes.

2. Crear una infraestructura de bajo costo para el desarrollo experimental de software distribuído.

El esquema de la red es el de un conjunto de nodos heterogéneos (computadores grandes, medianos y pequeños existentes en la Universidad) que a través de las Interfases se interconectan al compartir un medio de transmisión común (topología tipo bus).

La interfase es un dispositivo (un microcomputador) que continuamente está dispuesto a recibir "paquetes" de los nodos y a ponerlos a viajar por el medio de transmisión compartido a su(s) destino(s). Igualmente la Interfase está continuamente dispuesta a recibir paquetes del medio para dirigirlos al nodo, cuando sea necesario.

Este artículo comienza presentando en la sección 2 una visión general de la Interfase. Se ubica ésta dentro de la jerarquía de protocolos de comunicación y se recalca su importancia. La sección 3 describe con algún detalle los requerimientos mínimos de hardware para una Interfase realista (cuyo hardware esté al alcance de nuestro medio) en el caso de la red Uniandes. La sección 4 presenta el software diseñado para el manejo de la Interfase. Más detalles de este trabajo se encuentran en (ARAU80).

2. CONSIDERACIONES GENERALES

Cuando se desea interconectar dos máquinas normalmente es posible encontrar una solución fácil y realizable a corto plazo : se trata de usar lo ofrecido por las diferentes casas de los fabricantes. Si los equipos son homogéneos es muy sencillo lograr que cada uno "vea" al otro como si fuera una terminal. Si los equipos son heterogéneos es posible estandarizar la comunicación mediante el uso de emuladores como el 3780 de IBM (ofrecido por Texas, Burroughs, WANG, etc) ó 3270 de IBM. Obviamente este tipo de soluciones es aceptable cuando los requerimientos de comunicación son muy elementales. Ahora bien, si se desea conectar una nueva máquina, el problema es aún relativamente fácil de resolver porque bastaría colocar dos líneas más para comunicar el tercer computador con los existen

tes. Conectar un cuarto o más computadores implica pensar adicionalmente en cuáles de ellos se interconectarían con estos, ya que por un lado existe en cada máquina un límite físico de puertos de comunicación y por otro, conectarlos todos entre sí incrementa el costo de canales de comunicación (incluyendo el de modems si es el caso). Adicionalmente es necesario comenzar a introducir software confiable de enrutamiento y evitar que la disponibilidad del sistema dependa de la disponibilidad de unas pocas máquinas. Si seguimos expandiendo la red, estos problemas se tornan aún más complejos (y típicamente en redes de amplia cobertura geográfica se resuelven con hardware y software de comunicación costoso).

En un ambiente de red local las topologías de canal común (v.g. bus o anillo) evitan que sucedan los problemas anteriormente mencionados. La red Uniandes usa un sistema de comunicación local soportado por una topología de tipo bus. Específicamente esta concepción de red ofrece varias ventajas :

- Se crea un ambiente fácil de expandir, porque es sencillo añadir nuevos nodos sin preocuparse por adquirir nuevas líneas, gastar puertos de comunicación y enrutar.
- Se presta esta topología para desarrollar un sistema confiable en el sentido de que la caída de un nodo o Interfase no afecta el funcionamiento del resto de la red.
- Permite tener un protocolo de acceso (método usado para compartir ordenadamente el medio de transmisión) sencillo y distribuído.

2.1. CARACTERISTICAS DE LA INTERFASE

El hardware de comunicación de una red local tiene como objetivo dar un alto rendimiento a bajo costo. Como en este ambiente típicamente se desean interconectar minicomputadores y microcomputadores de bajo costo es necesario hacer tan barato como sea posible el hardware de la Interfase. Es claro que sería más eficiente realizar físicamente la Interfase como un minicomputador de comunicaciónes ("front-end") pero este tipo de solución haría que fácilmente el costo de la Interfase fuera superior al costo de algunos nodos utilizados. Es esta la principal razón por la cual se implantan las Interfases como microcomputadores.

En la red Uniandes cada Interfase asociada con una máquina provee el control de la transmisión y recepción de paquetes de bits. Tal función involucra el chequeo de errores causados por problemas en el medio de transmisión y que inciden sobre los paquetes que por él viajan. Adicionalmente la Interfase reconoce direcciones con el fin de determinar cuál es el destino de los paquetes de bits que le llegan (todos los paquetes llegan a cada Interfase). Existe, también, un mecanismo encargado de detectar y resolver las colisiones que pueden ocurrir durante la transmisión de los paquetes.

2.2. LOCALIZACION DE LA INTERFASE

La figura 1 muestra en forma general el diagrama de la red Uniandes, y en ella se aprecia la ubicación física de las Interfases, las cuales están conceptualmente divididas en dos : una parte orientada a la línea (comple

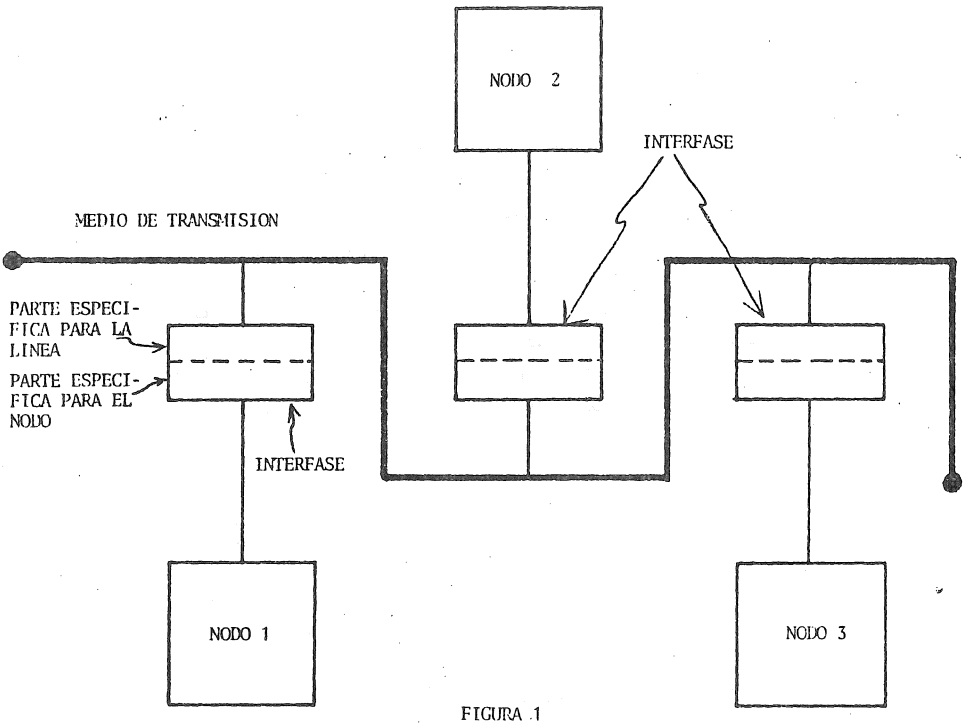


FIGURA 1
ESQUEMA GENERAL

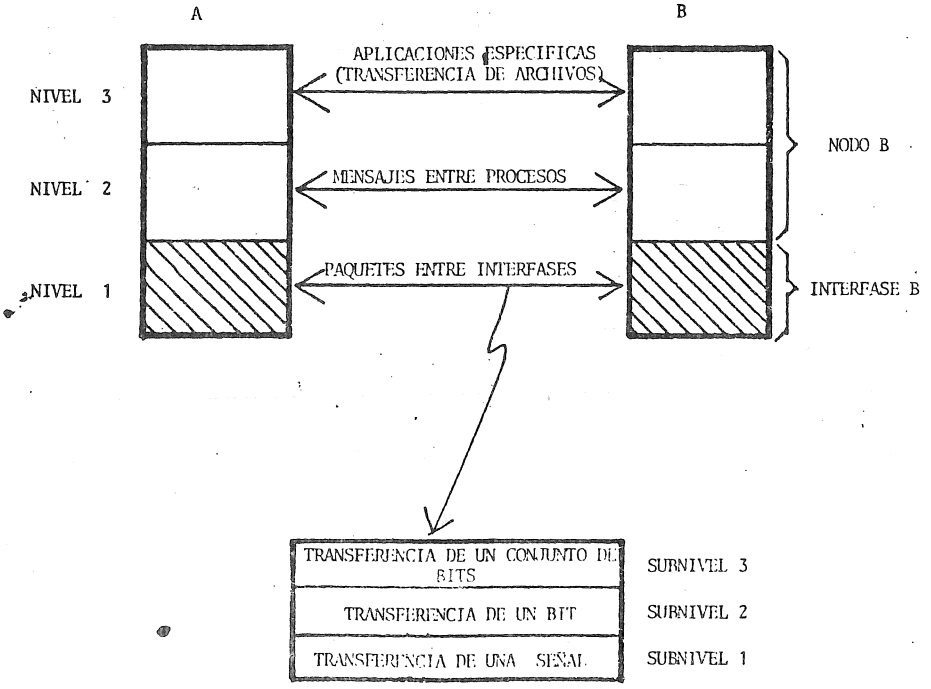


FIGURA 2
LOCALIZACION

tamente estándar), que realiza todas las funciones de control que son requeridas por ésta, y una parte orientada al nodo (varía en detalle según el nodo), que controla el intercambio de paquetes entre el nodo y la parte orientada a la línea.

Dentro de la jerarquía de protocolos de comunicación (ver figura 2) el intercambio de mensajes entre las Interfases pertenece al nivel más bajo (nivel 1 en la figura 2). Sobre éste se encuentran dos niveles más : el de mensajes entre procesos y el de aplicaciones específicas (v.g. transferencia de archivos (GAIT80), sistemas de archivos distribuídos (SANC80)).

Con esta concepción de diseño es posible hacer modular la comunicación nodo a nodo, de tal manera que cambios realizados en un nivel no alteren los demás, siempre y cuando se preserven las mismas Interfases. El nivel 1 se divide, a su vez, en tres subniveles : 1) transferencia de señales, en donde se resuelve el problema de reconocimiento de señales desde el punto de vista eléctrico, 2) transferencia de bits, en donde se resuelve el problema de reconocimiento de bits (las señales del subnivel anterior codifican bits), y 3) transferencia de conjuntos de bits (paquetes). Este artículo se refiere básicamente a los problemas de los subniveles dos y tres (una buena discusión sobre estructura y niveles de protocolos se encuentra en (PARD79), (BOGG80)).

2.3. IMPORTANCIA DE LA INTERFASE

Vale la pena recalcar la importancia de incluír una Interfase en la red : 1) Se estandariza la comunicación entre los diferentes nodos colgados a la red (el hecho de tener una Interfase estandar facilita la expansión de la red, puesto que "colgarle" un nuevo nodo no trae mayores complicaciones), y 2) se libera, hasta cierto punto, al procesador de los trabajos relacionados con comunicación (teniendo en cuenta las capacidades de una Interfase poco costosa).

2.4. TRABAJOS RELACIONADOS

Una de las principales motivaciones en el proyecto de nuestra red es "embarcarnos" en el desarrollo de una tecnología reciente y sobre la cual se está trabajando en muchos sitios. Creemos que hay varias ventajas en este ejercicio : 1) cuando esta tecnología llegue en forma de "producto" a nuestro país podremos evaluarlas al nivel que queramos su calidad técnica (algo que raras veces sucede en tecnología de computadores ya que invertimos esfuerzos en sólo entender lo que nos venden, y peor aún, a veces sólo después de que nos lo han vendido), 2) aunque existan redes "mejores" la experiencia en el diseño e implantación de estos sitemas es invaluable.

La red pionera en sistemas de comunicación local es DCS (FARB72), desarrollada en la Universidad de California, Irvine, la cual interconecta varios nodos heterogéneos en forma de anillo. Sin embargo los adelantos más interesantes en redes locales han sucedido en los últimos 5 años. ETHERNET (METC80, CRAN 80, THAC79), desarrollada en Xerox Palo Alto Research Center, es tal vez la red local más popular hoy en día (aparentemente será un producto conjunto de XEROX/DIGITAL/INTEL) que

interconecta mediante un bus nodos homogéneos. La Interfase es parte del nodo mismo e implanta un protocolo de contención. Otra red con igual topología y protocolo es NBSNET (CARP80); sin embargo la Interfase de ésta es un microcomputador un tanto sofisticado. Otro producto existente en el mercado desde hace varios años es el HYPERCHANNEL (DONN79) el cuan interconecta componentes, generalmente de la compañía CDC, a través de un bus usando un protocolo de contención. Prime Computers, ofrece RINGNET (GORD80) como un producto el cual usa topología tipo anillo y un protocolo que circula un mensaje especial ("token") para controlar el acceso al anillo. Zilog anunció una red similar al ETHERNET llamada Z-NET. Finalmente existen compañías no-fabricantes de computadores que venden o cables, o Interfases muy elementales, o ambos.

Muchas universidades y centros de investigación han desarrollado o están desarrollando redes locales. Interesantes entre éstas, el proyecto del MIT de su LCSNET (CLAR78, SALT80) y el de Ohio State con su DDLCN.

3. HARDWARE DE LA INTERFASE

Recordemos que la tarea principal de la Interfase consiste en recibir paquetes de bits del nodo para enviarlos a su destino vía la línea de transmisión, además de transportar los que por ella vienen con destino al nodo. Para realizar esta labor, es necesario comunicar a la Interfase con la línea y con el nodo. En la figura 3 se muestra el hardware mínimo que necesita la Interfase para cumplir con sus requisitos de funcionamiento : a) una unidad de microprocesamiento con, por lo menos, cinco puertos de entrada y salida para conectar la Interfase con el nodo, conectar la Interfase con la línea y conectar dos relojes externos (más adelante se explica su uso); además es deseable que el microprocesador posea dos niveles de interrupción (uno enmascarable y otro no enmascarable); b) memoria permanente y temporal, y c) una Interfase para comunicación con el nodo y otra para comunicación con la línea.

3.1. IDENTIFICACION Y SELECCION DE COMPONENTES

a) MICROPROCESADOR : se escogió el Motorola 6800 (MOTO76) por su disponibilidad y fácil uso (lo relevante del artículo no es mostrar una selección óptima de componentes). Este microprocesador tiene las siguientes características :

- bus de datos de 8 bits (bidireccional),
- capacidad de direccionamiento hasta 64 K bytes (bus de direcciones de 16 bits),
- 72 instrucciones,
- 7 modos de direccionamiento,
- interrupción enmascarable,
- interrupción no enmascarable,
- 6 registros internos, y
- perifericos que se referencian como posiciones de memoria.

b) MEMORIA PERMANENTE : se escogió la de tipo PROM (Programable Read Only Memory) para guardar los programas que manejan la Interfase

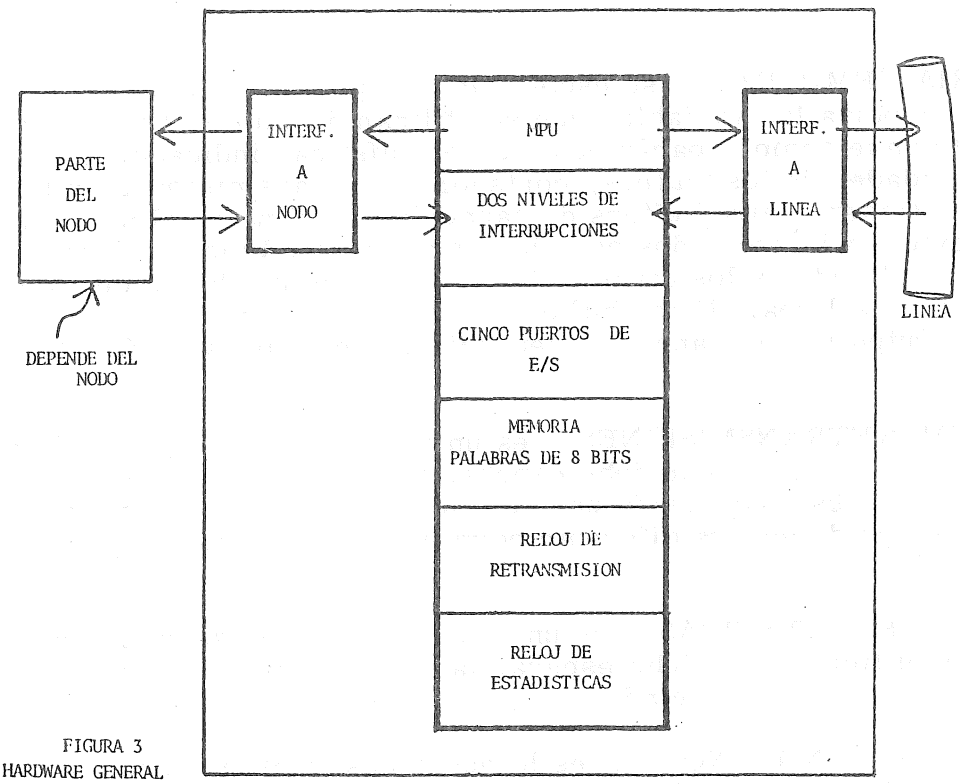


FIGURA 3
HARDWARE GENERAL

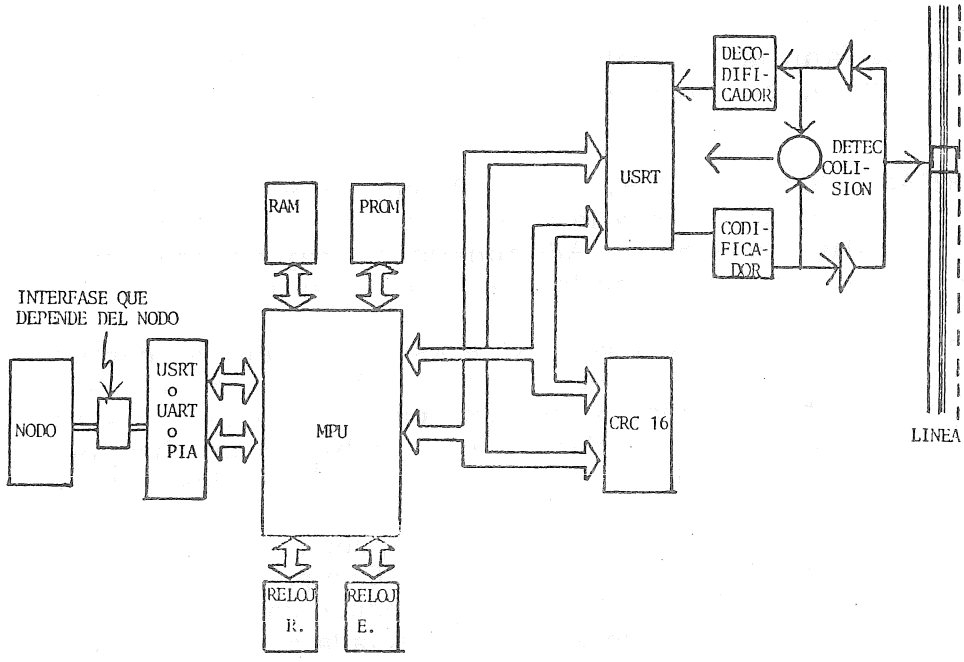


FIGURA 4
INTERCONEXION DE HARDWARE

algunos datos fijos (dirección del nodo y la dirección del nodo de estadísticas), y los apuntadores que proveen las direcciones de los programas que deben ser ejecutados ante la ocurrencia de una interrupción determinada.

c) MEMORIA TEMPORAL : se escogió la de tipo RAM (Random Access Memory). Soporta las variables que se utilizan durante el funcionamiento de la Interfase como : parámetros de estadística, indicadores del estado y prioridades de los buffers, contador de retransmisiones, y los buffers. Se tienen cuatro buffers de capacidad máxima de almacenamiento de 256 bytes cada uno : dos para los paquetes recibidos del nodo y transmitidos al nodo, y los otros para los paquetes recibidos del nodo y transmitidos a la línea. Estos buffers se usan en forma FIFO (First Input First Output) y se manejan independientemente para la línea y para el nodo.

d) RELOJ DE RETRANSMISIONES : es un reloj externo que se utiliza para contabilizar el tiempo que debe esperar la Interfase para retransmitir un mensaje después de ocurrida una colisión en la línea. Cada Interfase tendrá un tiempo de espera diferente porque, de no ser así, ocurriría de nuevo una colisión.

e) RELOJ DE ESTADISTICAS : es un reloj externo que se utiliza para contabilizar el tiempo que debe esperar la Interfase para ensamblar y enviar las estadísticas recolectadas.

f) INTERFASE CON EL NODO : es lo que necesita el microcomputador para comunicarse con el nodo y depende de las facilidades que provea el nodo. Por lo general, la comunicación ofrecida es en serie; si es asincrónica se utiliza un chip del tipo UART (Universal Asynchronous Receiver/Transmitter), o si es sincrónica se utiliza uno de tipo USRT (Universal Sychronous Receiver/Transmitter). En caso de que la comunicación sea en paralelo, la familia Motorola 6800 provee un chip llamado PIA (Peripheral Interface Adapter), el cual se encarga de la recepción y transmisión en paralelo.

g) INTERFASE CON LA LINEA : para la comunicación con la línea se necesita un chip que transfiera la información serial y sincrónicamente. Para ello se utiliza un USRT. Adicionalmente hay que identificar otros componentes que hacen parte de la Interfase con la línea :

- CHIP CRC-16 (Cyclic Redundancy Checksum): es el más confiable de todos los tipos de chequeadores de error en la transmisión de los paquetes. Agrega 16 bits al final del paquete cuando éste se transmite. Al recibirse el paquete, se calculan 16 bits, en base a una función polinomial y se comparan con los 16 bits de chequeo recibidos (WEIS79).

- DETECTOR DE COLISIONES : es una compuerta Or-Exclusivo que realiza la función de detección de colisiones. Compara los datos recibidos con los que están siendo transmitidos y produce una señal de colisión si hay alguna diferencia (esta señal está conectada a la entrada de la interrupción no enmascarable del microprocesador, dada su alta prioridad). Una colisión se produce cuando dos o más Interfases hacen acceso a la

línea al mismo tiempo (CRAN80).

- CODIFICADOR Y DECODIFICADOR DE SEÑALES : dependiendo del medio de transmisión que se utilice, se varía la clase de estos componentes para hacer compatibles los de señales que éste entiende con las que entiende la Interfase (CRAN80).

La figura 4 ilustra la interconexión de los componentes de hardware de la Interfase.

4. DISEÑO DE SOFTWARE

4.1. ESPECIFICACION

La descripción del software juega un papel muy importante en todos los pasos de su desarrollo. En esta sección se intenta explicar claramente y sin ambiguedades (i.e. mediante métodos formales), las funciones que debe desempeñar el software de la Interfase de la red.

La arquitectura de comunicación de un sistema distribuído, para su mejor entendimiento y para lograr buena modularidad, debe estar estructurada como una jerarquía de diferentes niveles. Cada nivel, provee un conjunto particular de servicios a sus usuarios superiores. Desde este punto de vista, el nivel puede verse como una caja negra o máquina que permite un cierto conjunto de interacciones con otros usuarios. A cada usuario debe interesarle la naturaleza del servicio provisto, pero no la forma en que el módulo lo implementa.

Esta descripción del comportamiento de Entrada/Salida de cada nivel es lo que constituye una especificación del servicio prestado por éste.

De la misma forma que es necesario establecer una estructura jerárquica entre los diferentes niveles del software de comunicación, dentro de cada nivel resulta muy útil establecer módulos claramente diferenciados, y especificarlos de acuerdo a las convenciones anteriormente descritas con el fin de obtener una visión completa de las funciones que estos desempeñarán.

El método que será usado en este artículo para especificar formalmente el software serán los "diagrmas de estados" (DANT80), por medio de los cuales es posible controlar y entender todos los posibles eventos concurrentes que pueden suceder en un momento dado dentro de la Interfase.

SECUENCIA GENERAL DE EVENTOS.
Existen claramente definidas tres funciones generales que desempeña la Interfase :

- Servir de intermediario para el transporte de paquetes del nodo hacia la línea.

- Servir de intermediario para el envío de paquetes de la línea hacia el nodo.

- Ensamblar paquetes de estadísticas y enviarlos con destino al nodo encargado de su recolección.

La figura 5 muestra la secuencia general de eventos cuando la Interfase se encuentra transportando un paquete desde el nodo hacia la línea. Inicialmente la Interfase está ociosa (no tiene ningún trabajo por realizar) en espera de algún pedido que deba ser atendido. Estos pedidos se efectúan mediante interrupciones. Si la interrupción es causada por un paquete que viene del nodo, la interfase debe disponerse a recibirlo. Una vez se ha detectado su final, es necesario transmitirlo a su destino vía la línea de transmisión; es este momento es cuando se da el control al protocolo, que se encargará de efectuar confiablemente dicha transmisión. De este nuevo estado (protocolo) se sale cuando se detecta el final del paquete que está siendo transmitido, quedando la Interfase nuevamente ociosa.

La segunda función, ilustrada en la figura 6, se inicia cuando llega un pedido (interrupción) por parte de la línea, indicando que hay un paquete que debe ser recibido y que posiblemente se dirige hacia el nodo asociado con esa Interfase. Si este es el caso, es necesario recibir completamente el paquete entrante y analizar su dirección; cuando ésta concuerde con la asignada a tal nodo, el paquete es pasado a él y la Interfase se contrará ociosa. Si la dirección no concuerda, se pasa directamente al estado ocioso en espera de un nuevo pedido.

Finalmente, la función (figura 7) de la Interfase se relaciona con el ensamblaje y envío de las estadísticas que han sido recolectadas durante la ejecución de las demás funciones. Se parte del estado ocioso; cuando llega una interrupción del reloj de estadísticas, es necesario comenzar a ensamblarlas. Una vez se ha terminado, deben ser enviadas al nodo o a la línea, de acuerdo a la dirección del nodo encargado de recolectarlas.

Los diagramas de estados propuestos en las figuras 5, 6 y 7 describen el software que debe ser implantado en la Interfase, pero hacen indivisibles cada una de las funciones anteriormente explicadas. Sin embargo, el hecho de recibir un paquete del nodo o de la línea no debe implicar que sea enviado inmediatamente a su destino, ya que podría dejar de recibirse otros que vinieran seguidamente, es decir, debe haber suficiente independencia entre la recepción y la transmisión de un paquete, con el fin de poder atender recepciones consecutivas de éstos, sin dañar lo ya recibido (sólo retrasando su transmisión).

Para poder diseñar, entonces, un software que tenga capacidad de atender eventos concurrentes, las tres funciones anteriores serán implantadas por los módulos :

- Recepción de la línea;
- Transmisión al nodo;
- Recepción del nodo;
- Protocolo;
- Ensamblaje de estadísticas.

Estos módulos no serán interrumpibles; es decir, una vez que se han comenzado a ejecutar, terminarán su función a menos que una falla los

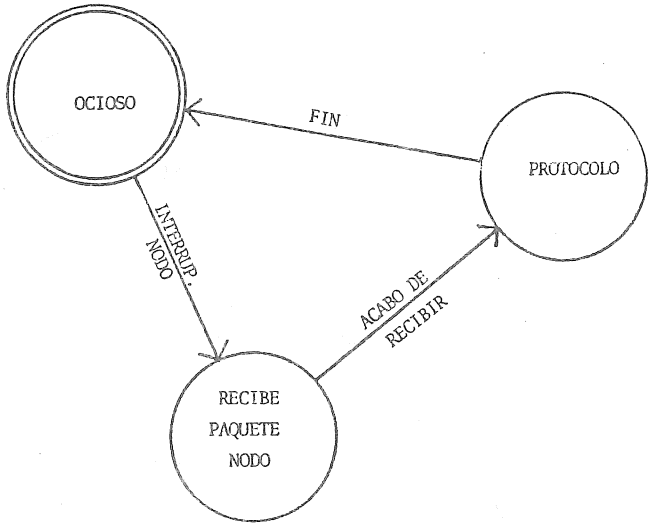


FIGURA 5
TRANSMISION A LINEA

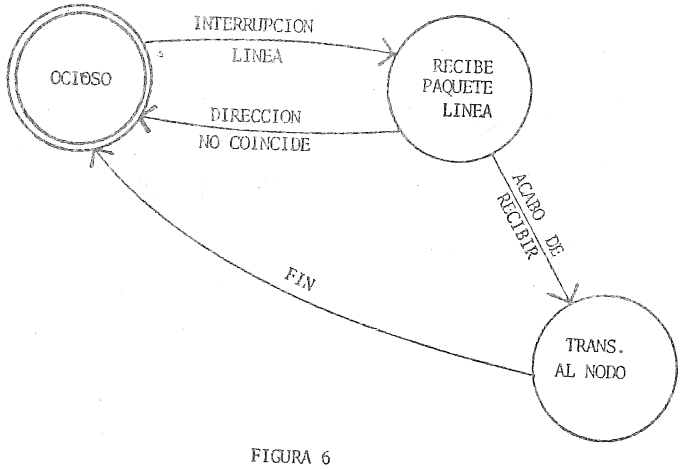


FIGURA 6
RECEPCION DE LINEA

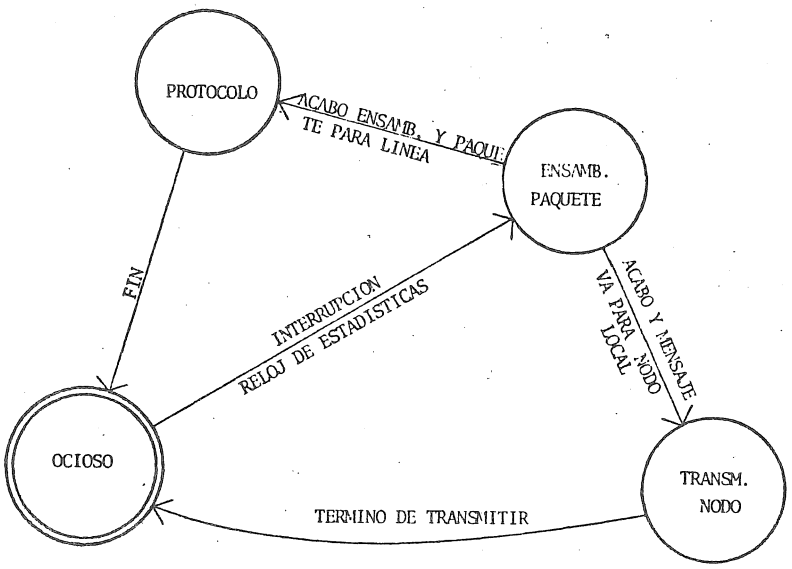


FIGURA 7
ENVIO DE ESTADISTICAS

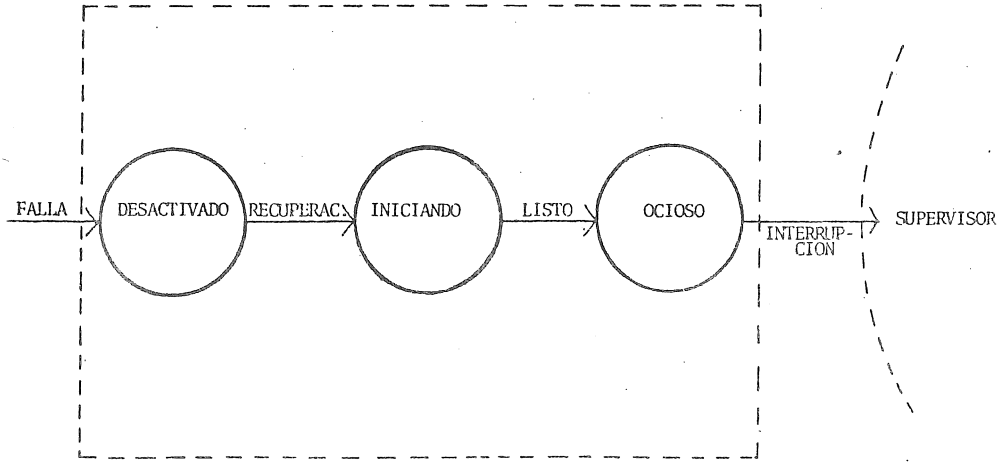


FIGURA 8
MODULO DE INICIACION

suspenda (toda interrupción que se produzca durante su ejecución, será diferida). En resumen, estos módulos con uno adicional de iniciación hacen lo siguiente :

- MODULO DE INICIACION : Este módulo tiene a su cargo iniciar la Interfase cuando se recupera después de una falla, o simplemente cuando es puesta en funcionamiento.

- MODULO DE RECEPCION DE LA LINEA : Se encarga de resolver los problemas relacionados con la recepción de los paquetes de bits provenientes de la línea (chequeo de errores en la información recibida, control de dirección).

- MODULO DE TRANSMISION AL NODO : Controla la transmisión, hacia el nodo, de los paquetes recibidos correctamente por el módulo de recepción de la línea. Este control varía en algunos detalles dependiendo del tipo del nodo al cual está atada la Interfase.

- MODULO DE RECEPCION DEL NODO : Recibe los paquetes provenientes del nodo. Al igual que el módulo anterior éste depende de las facilidades que presente el nodo.

- MODULO PROTOCOLO : Controla la transmisión, hacia la línea, de los paquetes recibidos por el módulo de recepción del nodo. Este control consiste en lograr un acceso ordenado, eficiente y confiable al medio de transmisión compartido.

- MODULO DE ESTADISTICAS : Ensambla las estadísticas recolectadas durante un intervalo predeterminado de tiempo, para luego ser enviadas al nodo designado para que las procese y reporte. Este módulo provee información para medir la eficiencia de la red y es opcional; es decir , existe la posibilidad de iniciar o suspender remotamente la recolección y envío de estadísticas.

Todos estos módulos se relacionan entre sí, y es necesario sincronizarlos para resolver los problemas causados por la concurrencia entre la llegada de paquetes ya sea del nodo o de la línea y lo que en ese momento esté realizando la Interfase. Se debe controlar, también, la escritura y lectura sobre los buffers en los cuales se reciben y se mandan los paquetes tanto de la línea como del nodo. Se hace entonces necesaria la creación de un séptimo módulo que supervise todos los eventos (MODULO SUPERVISOR).

4.2. DESCRIPCION DETALLADA DE LOS MODULOS

MODULO INICIACION :
Como cualquier sistema, la Interfase necesita ser iniciada. Después de ocurrida una falla, o de un receso en su funcionamiento, se entra en el estado de desactividad (ver figura 8). Una vez en funcionamiento el proceso a ejecutar, estado "iniciando", consiste en :

- Iniciar las variables utilizadas : Los indicadores del estado de los buffers y los acumuladores de estadística (ver módulo) se colocan en cero señalando que éstos se encuentran vacíos; la variable que indica si se recolec

tan estadísticas se pone en "cierto".
- Colocar en "cierto" la variable que indica que la Interfase acaba de ponerse en funcionamiento y que luego será enviada al nodo que procesa las estadísticas de la red.
- Cargar el reloj de estadísticas con un tiempo bastante corto para enviar un paquete, que tenga como objetivo informarle al nodo que procesa las estadísticas de la red que la Interfase en cuestión está funcionando. Sin embargo, el paquete no lo envía directamente este módulo ya que hay otro módulo que normalmente envía los paquetes.

Una vez terminado el estado "iniciando" la Interfase se pasa al estado "ocioso" en donde permanece hasta que reciba una interrupción cualquiera.

MODULO DE RECEPCION DE LA LINEA (FIGURA 9) :
A este módulo se llega cuando se va a atender una interrupción causada por un paquete proveniente de la línea. Sus estados son los siguientes :

- Chequear disponibilidad de buffer :
  Durante este estado se observan los indicadores del estado de los buffers de recepción de la línea. Si existe por lo menos uno de ellos disponible se pasa al estado "recibiendo paquete"; si por el contrario, los dos buffers destinados a la recepción de la línea se encuentran ocupados, se transfiere el control al módulo supervisor.

- Recibiendo paquete :
  En este estado, la Interfase está dedicada exclusivamente a recibir, pasando caracter por caracter desde el periférico encargado de la recepción de la línea hasta el buffer que le fue asignado a tal paquete en la memoria. Al tiempo, el chip de CRC está efectuando sus cálculos con el fin de chequear posibles errores ocurridos al paquete en le medio de transmisión.

  Cuando se detecta el fin de texto, se pasa al estado de "chequear CRC". Si por el contrario, al cabo de un tiempo no se detecta un caracter de fin de texto, se pasa a "liberar buffer" bajo el evento de mensaje incompleto.

- Chequear CRC :
  En este momento el CRC ya tiene una respuesta acerca de la información recibida, por tanto lo único que la Interfase debe hacer es averiguar por el resultado del chequeo. Así, si CRC está errado se pasa al estado "chequear dirección", de lo contrario se pasa a "liberar buffer".

- Chequear dirección :
  En este estado se chequea el destino que lleva el paquete que se acaba de recibir. Existen tres (3) posibilidades :

  - Paquete para la Interfase;
  - Paquete para el nodo;
  - Paquete para el otro nodo.

  Si el paquete es para la Interfase, se pasa a un estado de "activar o desactivar estadísticas"; si es para su nodo local se asegura de hacerlo llegar a éste; y finalmente, si es para otro nodo, se pasa a "liberar buffer".

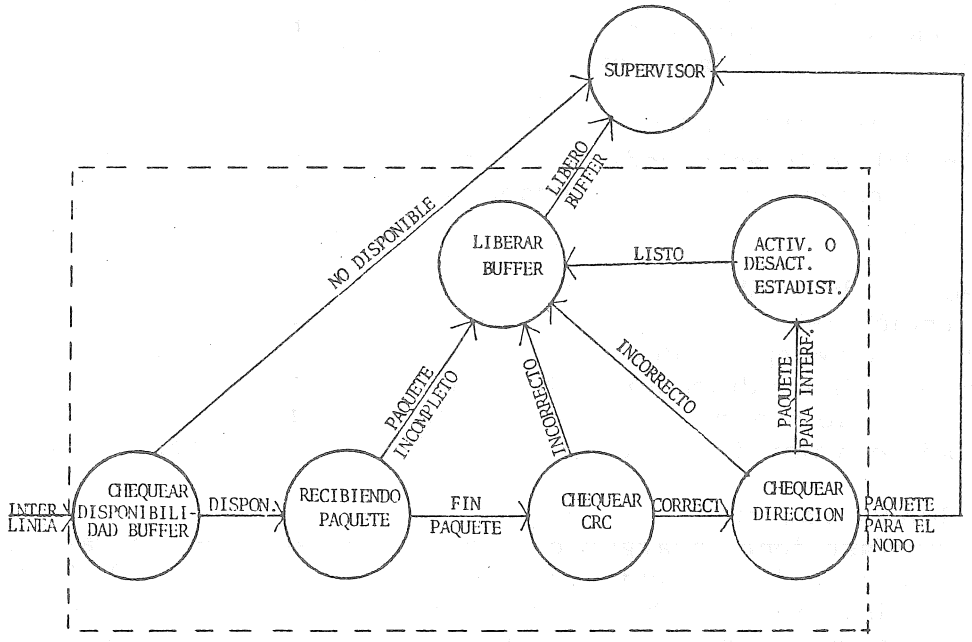


FIGURA 9
MODULO DE RECEPCION DE LA LINEA

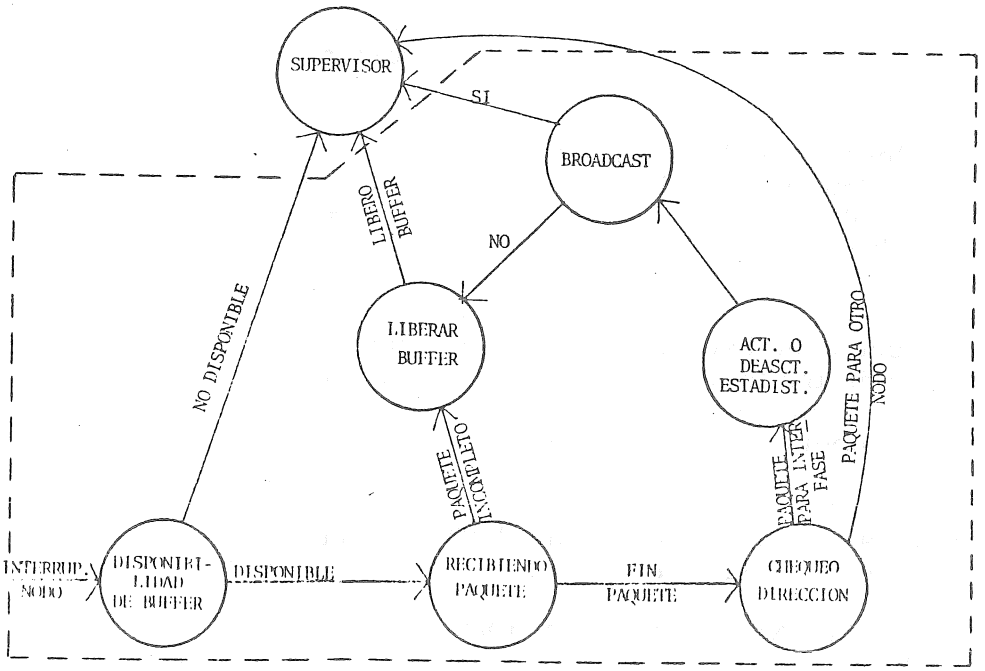


FIGURA 10
MODULO DE RECEPCION DEL NODO

- Liberar buffer :
  En este estado se debe alterar el indicador del buffer de recepción de la línea, el cual almacenó el paquete en cuestión. El nuevo valor señalará su disponibilidad para recibir cualquier otro paquete.

MODULO DE RECEPCION DEL NODO (FIGURA 10) :
Este módulo está encargado de atender la interrupción causada por un paquete proveniente del nodo asociado con la Interfase. Sus estados son los siguientes :

- Chequear disponibilidad de buffer :
  La función del módulo en este estado es chequear los indicadores de los buffers de recepción del nodo (transmisión a la línea) con el fin de saber si existe alguno disponible. Si es así, se pasa al estado "recibiendo paquete"; de lo contrario, se transfiere el control al supervisor, sin haber variado el valor del indicador, señalándolo como vacío.

- Recibiendo paquete :
  La Interfase debe encargarse de recibir caracter por caracter el paquete que el nodo le está enviando, pasándolo al buffer que le ha sido asignado en la memoria. Si durante esta recepción, es detectado el fin de texto, se pasa al estado de "chequeo de dirección". Si por el contrario, al cabo de un tiempo predeterminado, no se ha recibido un nuevo caracter, se asume que el paquete llegó incompleto y se va al estado "liberar buffer".

- Chequear dirección :
  En este estado debe chequearse la dirección de destino del paquete, con el fin de saber si se dirige a la Interfase o lleva como destino algún otro nodo. Si se dirige a otro nodo, el control es transferido al supervisor, pero si se dirige hacia la Interfase, se pasa al estado "activar/desactivar estadísticas".

- Activar/Desactivar Estadísticas :
  En éste se debe alterar la variable que indica si se debe activar o desactivar la recolección de estadísticas. Una vez se ha hecho esto, se pasa al estado "Broadcast".

- Broadcast :
  Lo único que debe hacerse ahora es averiguar si el paquete recibido es uno que se dirige también a todas las demás Interfases. Si no es así, debe darse el control al estado "liberar buffer"; de lo contrario se pasa al módulo supervisor.

- Liberar buffer :
  En éste se altera el indicador del buffer de recepción del nodo que contenía el paquete. Su valor lo señala como disponible.

MODULO DE TRANSMISION AL NODO :
A este módulo se entra cuando en el supervisor hay pendiente una transmisión al nodo y, según el orden de prioridades, hay que atenderla (Ver figura 11). Consta de dos estados :

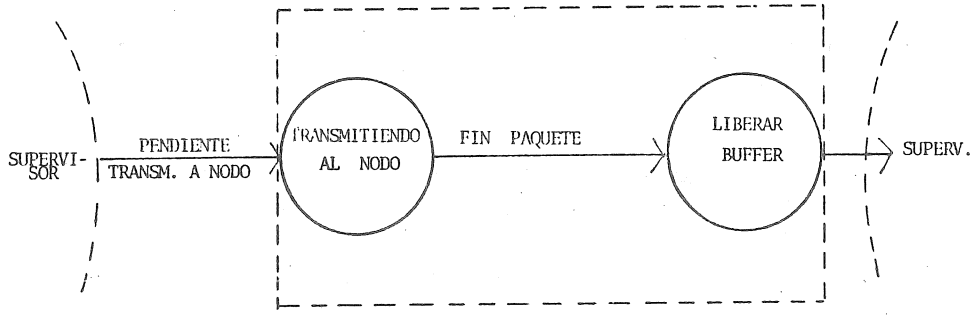


FIGURA 11
MODULO DE TRANSMISION AL NODO

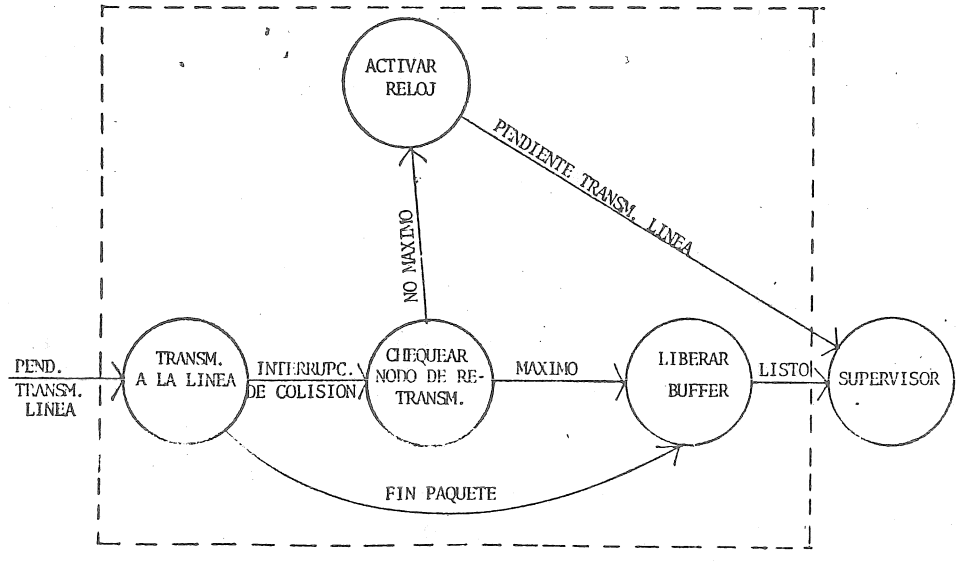


FIGURA 12
MODULO PROTOCOLO

- Transmisión al nodo :
  Se averigua, dependiendo de la prioridad, de cuál buffer se va a transmitir. Se coloca luego carácter por caracter en el periférico encargado de la transmisión física hasta que el contador de caracteres llegue al límite.

- Liberar buffer :
  El indicador del buffer que se transmitió se pone en vacío y se rearreglan las prioridades de los buffers. La transición para salir de este estado se hace hacie el módulo supervisor.

MODULO PROTOCOLO :
Los protocolos de acceso desde el punto de vista de cuándo se transmite en el canal compartido se dividen en tres grandes clases :

- Los de selección : una Interfase transmite cuando le llega s[illegible] (GORD80), (CLAR78).

- Los de contención : una Interfase transmite sin esperar turno o reservar, lo que implica resolver colisiones (METC80), (CLAR78).

- Los de reservación : una Interfase transmite en el intervalo de tiempo reservado para ella.

En la red se pretende usar un protocolo del tipo contención por ser estos sencillos (menos hardware y software) y por tener control distribuído. Se explica a continuación el diagrama de estados del protocolo de acceso utilizado. Consta de cuatro estados y se llega al inicial cuando en el supervisor hay pendiente una transmisión a la línea y, según el orden de prioridades, hay que atenderla (ver figura 12):

- Transmitiendo a la línea :
  Se averigua, dependiendo de la prioridad, de cuál buffer se va a transmitir. Se coloca entonces, caracter por caracter en el periférico encargado de la transmisión física hasta que el contador de caracteres llegue al límite o se detecta una colisión en la línea (produce una interrupción).

- Chequear número de retransmisiones :
  Se contabilizan cuántas retransmisiones se han hecho; se sale hacia dos estados dependiendo de si es máximo o no el número de retransmisiones.

- Activar reloj :
  Se activa el reloj de espera para retransmitir si el número no ha llegado al máximo Se va luego al supervisor con el pendiente de una transmisión pero con las demás transmisiones bloqueadas hasta que se acabe el tiempo ya iniciado.

- Liberar buffer :
  El indicador del buffer que se transmitió se pone en vacío y se arreglan las prioridades de los buffers. Se llega a este estado cuando se acabó de transmitir o cuando se llega al límite en retransmisiones. En caso de llegar al límite no se hace nada, y el problema debe ser resuelto por un protocolo de más alto nivel. Al salir del estado se va para el supervisor.

MODULO DE ESTADISTICAS :
Es importante hacer una recolección periódica de parámetros que midan en una u otra forma el rendimiento ya que se ayuda a :

1. detectar y corregir fallas de operación;
2. detectar y corregir errores de diseño, y
3. medir el rendimiento para afinar el sistema.

Los parámetros que se pueden medir son los siguientes : número de transmisiones, número de retransmisiones, longitud promedio de los mensajes, número de mensajes recibidos, número de mensajes con errores, número de mensajes que van para el nodo, número de mensajes rechazados por falta de buffer del nodo, número de mensajes rechazados por falta de buffer de la línea, mensajes incompletos, y caídas. De estos parámetros se pueden deducir otros como, por ejemplo, tiempo que dura ociosa la Interfase, etc.

Las estadísticas se recolectan durante el funcionamiento normal de la Interfase; cada intervalo de tiempo T, el reloj de estadísticas interrumpirá para pedir el envío de los datos recolectados. Con anterioridad se ha definido un nodo, en el cual se centralizan las estadísticas de la red para ser procesadas y reportadas. Se tiene de esta forma una visión global del funcionamiento de la red permanente (HEAR72).

La recolección de estadísticas y el envío para su procesamiento es opcional. En caso de que se desee o no hacerlo, se manda desde el nodo de estadísticas un mensaje a la(s) Interfase(s) informándole(s) al respecto. Cabe anotar, que al iniciarse la Interfase, se supone que está recolectando estadísticas.

A continuación se explica el diagrama de estados del ensamblaje de estadísticas (ver figura 13). Cuando sucede una interrupción del reloj de estadísticas y se atiende, no se ensambla de una vez sino que se avisa al supervisor que está pendiente el envío. Se entra, entonces, al módulo cuando hay pendiente un envío y se atiende. El módulo consta de cinco estados :

- Dirección nodo estadísticas :
  Dependiendo de la dirección del nodo de estadísticas, el mensaje se transmite hacia la línea o hacia el nodo de la Interfase en cuestión.

- Disponibilidad de buffer de la línea :
  Si la dirección del nodo de estadísticas no es la misma de la Interfase se pregunta si hay buffer disponible para mandar hacia la línea. Si no hay, se vuelve al supervisor dejando pendiente el envío; si hay buffer, se continúa con el estado "ensamblar para línea".

- Ensamblar para línea :
  Se ensambla el paquete que contendrá la información de las estadísticas recolectadas en uno de los buffers designados para transmitir a la línea. Para el ensamblaje se sigue el formato establecido (ver apéndice). De aquí se sale hacia el supervisor con una transmisión de la línea pendiente.

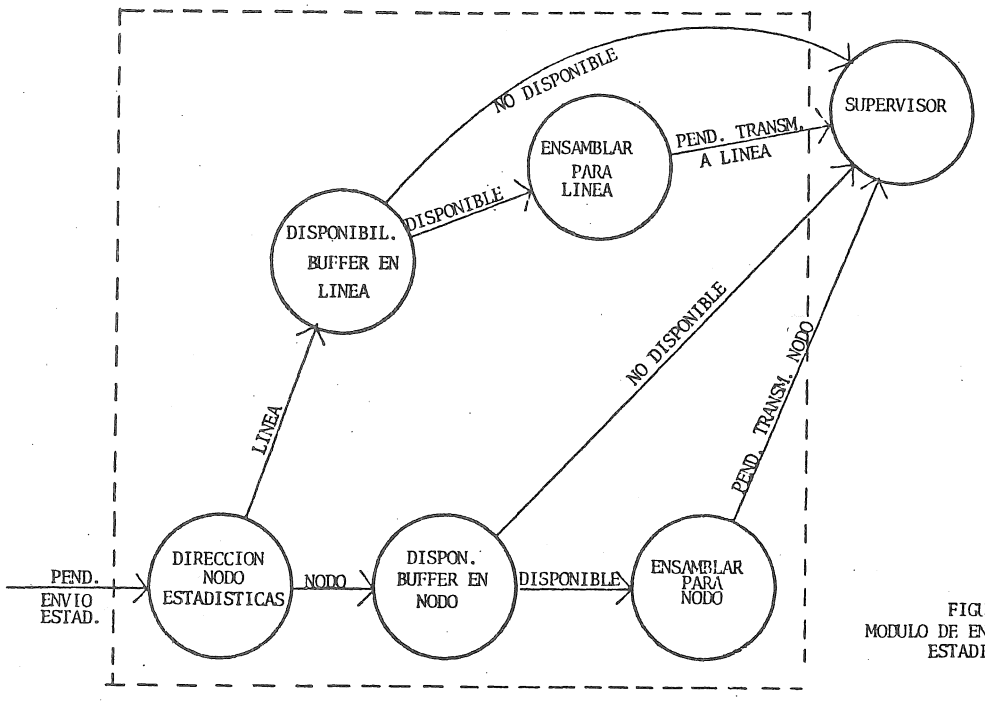


FIGURA 13
MODULO DE ENSAMBLAJE DE ESTADISTICAS

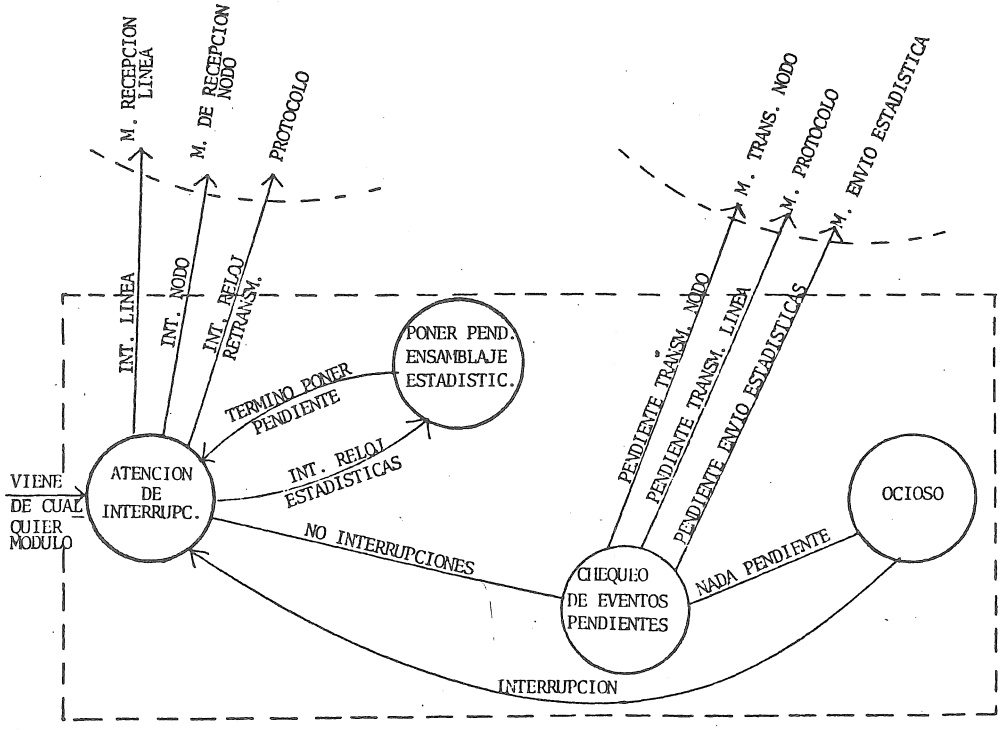


FIGURA 14
MODULO SUPERVISOR

- Disponibilidad de buffer del nodo :
  Se entra a este estado cuando la dirección del nodo de estadísticas es igual a la del nodo colgado a la Interfase en cuestión. Se pregunta si hay buffer disponible para mandar hacia el nodo; si no lo hay, se vuelve al supervisor quedando pendiente el envío; si lo hay, se continúa con el estado ensamblar para el nodo.

- Ensamblar para el nodo :
  Es similar al estado "ensamblar para la línea" teniendo en cuenta que se transmite hacia el nodo.

MODULO SUPERVISOR (VER FIGURA 14) :
Este módulo, de acuerdo a lo que su nombre indica, tiene como función coordinar la ejecución de las diferentes rutinas que se encargan de atender las interrupciones o ejecutar las tareas pendientes (envíos de mensajes o ensamblaje de estadísticas). (ESTU80).

Al módulo supervisor se entra viniendo de cualquier otro (ver módulos) cuando termina su función o cuando por algún motivo no la puede realizar.

Este módulo se compone de los siguientes estados :

- Atención de interrupciones :
  La función del supervisor, en este estado, es "monitorear" las posibles causas de interrupción :

  a. Interrupción de la línea, en cuyo caso se transfiere el control al módulo de recepción de la línea.
  b. Interrupción del nodo, ante este tipo de pedido debe tomar control el módulo de recepción del nodo.
  c. Interrupción del reloj de Retransmisiones, en cuyo caso debe ser atendida por el módulo protocolo.
  d. Interrupción del reloj de estadísticas, la acción a seguir, es únicamente pasar el control al estado de "poner pendiente ensamblaje de estadísticas".

  Si no hay interrupciones se pasa a "chequeo eventos pendientes".

- Poner pendiente ensamblaje de estadísticas :
  Después de recibir una interrupción del reloj de estadísticas no se va a hacer directamente su ensamblaje; más bien se cambia el valor del indicador que lo señala como pendiente. La razón de hacerlo así, es que sería preferible ensamblar las estadísticas en el momento en que no hubiera interrupciones por atender, ya que su envío no resulta ser prioritario.

- Chequeo eventos pendientes :
  En este momento, el supervisor chequea y atiende los eventos que están pendientes en espera de atención. Si hay pendiente una transmisión al nodo o a la línea, debe dársele el control a los módulos encargados de atenderla. Si es un ensamblaje de estadísticas lo que se tiene en espera, debe transferirse el control al módulo que las ensambla. Cuando no hay eventos pendientes, la Interfase entra en un estado "ocioso", hasta ser nuevamente interrumpida.

CONCLUSIONES

En este artículo se han descrito algunos de los principales componentes de la Interfase que conecta un nodo a una red local. Se hizo énfasis en las partes referenciadas al diseño mismo de este tipo de hardware basado en facilidades provistas por partes de relativa alcanzabilidad en nuestro medio. Bien se habría podido tratar de diseñar utópicas conexiones de eficientes módulos completamente inalcanzables. Por eso, fue nuestro parecer restringido y bastante realista.

Se quizo además, evitar detalles no relevantes en la descripción, con el fin de obtener mayor claridad en el diseño.

El software se diseñó de la forma más sencilla posible sin perder eficiencia. Todos los módulos en lenguaje de alto nivel se encuentran en (ARAU80).

La conclusión más importante es que no es muy complicado diseñar este tipo de hardware y software y que por lo tanto es factible de emprender este tipo de proyectos en nuestros países.

AGRADECIMIENTOS

Queremos hacer un reconocimiento especial al Doctor Antonio García, Jefe del Departamento de Ingeniería Eléctrica de la Universidad de Los Andes, por su colaboración con algunas sugerencias técnicas que han sido valiosas para el diseño de la parte hardware de la Interfase.

REFERENCIAS

(ANDER79). Steven C. Andersen, "A Serial Data Bus Control Method", Computer Networks 3, North-Holland Publishing Company, 1.979, pp. 447-457.

(ARAU80). Leonardo Araújo R. y Jairo Fernández S., Diseño de la Interfase para la Red Local Uniandes, Proyecto de Grado, Universidad de Los Andes, 1.980.

(BOGG80). David R. Boggs, "Pup: An Internetwork Architecture", IEEE Transactions on Comunications, Vol. Com. 28, No. 4, Abril 1.980, pp. 612-624.

(CARP80). Robert J. Carpenter y J. Sokol, "Serving Users with a Local Area Network", Computer Networks, Vol. 4, No. 5, North-Holland Publishing Co., Octubre/Noviembre 1.980, pp. 209-214.

(CLAR78). David D. Clark, et. al., "An Introduction to Local Area Networks", Proceedings of the IEEE, Vol 66, No. 11, Noviembre 1.978, pp. 1497-1516.

(CRAN80). Ronald C. Crane y Edward A. Taft, "Practical Considerations in Ethernet Local Network Design", Xerox Corporation, Febrero 1.980.

(DANT80). André A. S. Danthine, "Protocol Representation with Finite State Models, Vol. Com. 28, No. 4, Abril 1.980.

(DONN79). James E. Donnelley y Jeffry W. Yeh, "Interaction Between Protocol Levels in a Prioritized CMSA Broadcast Network", Computer Networks3, North-Holland Publishing Company, 1.979, pp. 9-23.

(ESTU80). Jacky Estublier, Curso de Sistemas Operacionales, Universidad de Los Andes, Vol. 1, Enero 1.980.

(FARB72). D. J. Farber y K. C. Larson, "The System Architecture of the Distributed Computer System - The Comunication System" Symposium on Computer Networks, Polytechnic Institute of Brooklyn, Abril 1.972.

(GAIT80). Laureano Gaitán y Rodrigo Paredes, Diseño de Protocolos para la Red Local Uniandes, Proyecto de Grado, Universidad de Los Andes, 1.980.

(GORD80). R. L. Gordon, et. al., "Ringnet : A Packet Switched Network with Descentralized Control", Computer Networks 3, North-Holland Publishing Company, 1.980, pp. 373-379.

(HEAR72). F. E. Heart, et. al., "The Interface Message Processor for the ARPA Computer Network", Computer Communications, 1.972.

(HSI 80). Peter Hsi y Tsvi Lissack, "Local Networks' Consensus : High Speed", Data Communications, Network Analysis Corporation, Diciembre 1.980, pp. 56-66.

(METC80). Robert M. Metcalfe y David R. Boogs, "Ethernet : Distributed Packet Switching for Local Computer Networks", Xerox Corporation, 1.980.

(MOTO76). Motorola, Benchmark Family for Microcomputer Systems M6800 Systems References and Data Sheets. 1.976.

(PARD79). Roberto Pardo S., "Estructura de Protocolos en Redes de Computadores", Colombia Electrónica, Nos. 7 y 8, 1.979, pp. 23-45.

(SANC80). Edilberto Sánchez y Edgar Ruiz, Diseño de un Sistema de Archivos Distribuído para la Red Local Uniandes, Proyecto de Grado, Universidad de Los Andes, 1.980.

(SALT80). J. H. Saltzer y K. T. Pogram, "A Star-Shaped Ring Network with High Maintainability", Computer Networks, Vol. 4, No. 5, Noth-Holland Publishing Co., Octubre/Noviembre 1.980, PP. 239-244.

(THAC79). C.P. Thacker, et. al., "Alto : A personal Computer", Xerox Corporation, 1.979.

(WEIS79). Alan J. Weissberger, "An LSI Implementation of an Intelligent CRC Computer and Programmable Character Comparator", IEEE Transactions on Computers, Vol. C-29, No. 2 , Febrero 1.980, pp. 116-124.

APENDICE A

Formato del paquete en la Red Uniandes :

El paquete comienza con un bit de sincronización cuya llegada capacita al receptor para adquirir la fase de bit (sincroniza la recepción). En seguida se tienen dos campos de dirección de 8 bits, la dirección de destino hacia el cual se dirige el paquete y la dirección del nodo en donde se originó. La dirección de destino posee sólo 7 bits para tal efecto y su octavo bit especifica si el paquete recibido es para la Interfase o para el nodo colgado a ella (uno o cero según el caso). El siguiente byte contiene la longitud del paquete (no incluye el bit de sincronización ni los bytes de CRC). Posteriormente se encuentra el campo de texto cuya longitud varía entre Ø y 253 bytes.

Finalmente, se cuenta con 16 bits de CRC (Cyclical Redundancy Check). El bit de sincronización y el CRC son generados y eliminados por medio del hardware de la Interfase; toda la información restante es transferida a 0 de la memoria de la Interfase.

El campo de texto es transparente para la Interfase y va dirigido a los protocolos del nivel superior.

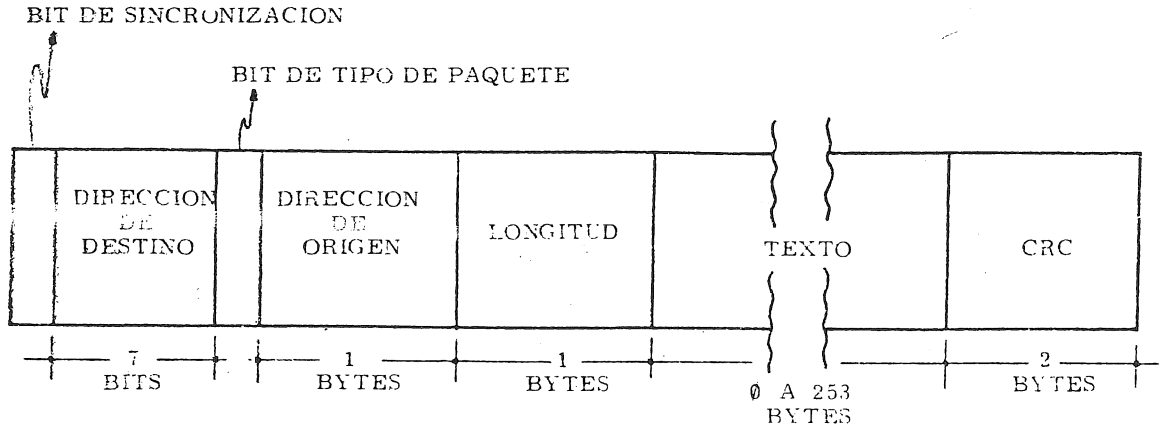


FIGURA 15
FORMATO DEL PAQUETE

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

DISEÑO DE PROTOCOLOS PARA LA RED LOCAL UNIANDES

Rodrigo Paredes
Laureano A. Gaitán
Roberto Pardo

Departamento de Ingeniería de Sistemas y Computación
Universidad de Los Andes
Apartado Aéreo 4976, Bogotá - Colombia.

OBJETIVO

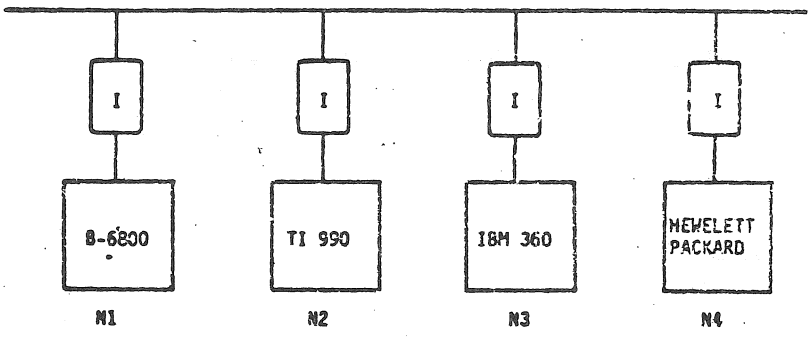
Este artículo describe las principales decisiones de diseño realizadas en el desarrollo de protocolos para la red local UNIANDES. Específicamente se concentra en dos tipos de protocolos : el de comunicación entre procesos o de transporte (PT) y el de transferencia de archivos (PTA).

La descripción de cada protocolo comprende : a) las funciones que debe realizar, b) las suposiciones, c) una especificación, d) la interfase que define, y e) los módulos necesarios para su implantación. Varios ejemplos y sugerencias sobre implantación complementan estas descripciones.

TERMINOS CLAVES : Redes locales, Protocolos de Alto Nivel, Implantación de Protocolos.

1. INTRODUCCION.

La "RED UNIANDES" es una red local experimental - actualmente en fase de diseño - nació como un esfuerzo conjunto entre los Departamentos de Ingeniería de Sistemas y Computación e Ingeniería Eléctrica de la Universidad de los Andes. El esquema general de la red es el de un conjunto de nodos hetereogéneos (computadores grandes, medianos y pequeños existentes en la Universidad) que a través de interfases se interconectan al compartir un medio de transmisión común (topología tipo bus) (Ver figura 1).



I: INTERFASE (M 6800)
Ni: NODO I
FIGURA 1 - Topología Red UNIANDES.

Durante el desarrollo del proyecto se han identificado varios problemas para atacar: el medio de transmisión, el hardware de la interfase, el software de comunicación residente en las interfases y en los nodos, software distribuído, etc. Los componentes más importantes del software de comunicación son los protocolos, o sea las reglas y convenciones necesarias para intercambiar información entre entidades (interfases, procesos). El objetivo de este artículo es describir las decisiones de diseño de dos tipos de protocolos de la red y que residen en los nodos: el de transporte (PT) y el de transferencia de archivos (PTA).

Tipicamente los protocolos de una red forman una jerarquía bien definida de servicios, donde los servicios de un nivel se 'aumentan' mediante un protocolo del nivel siguiente hacia arriba. En la sección 2 se describe la arquitectura de protocolos de la red.

Para describir el diseño de cada protocolo, secciones 3 y 4, se utiliza la siguiente metodología: primero se describen las funciones que deben hacer y los supuestos en cada caso. A continuación se hace una especificación, es decir se describe sin ambiguedades lo que el protocolo debe hacer. Tam

bién en esta parte se definen las primitivas (interfase) externas que 've' el nivel de más arriba para cada protocolo. Finalmente se procede a describir detalladamente los módulos necesarios para implantar cada tipo de protocolo. Varias sugerencias sobre detalles de implantación se ilustran en el artículo.

2. ARQUITECTURA RED UNIANDES.

En la figura 2 se muestra los principales niveles conceptuales que constituyen una arquitectura. Los niveles son importantes puesto que además de aislar y modularizar funciones específicas, sierven de soporte para el desarrollo de niveles superiores.

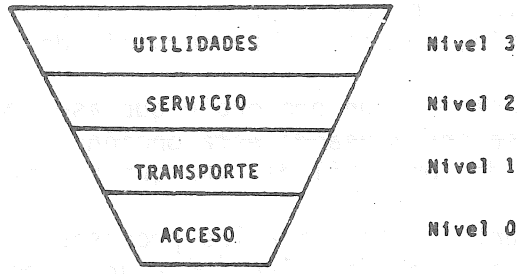


FIGURA 2 - Niveles conceptuales de una Arquitectura

El nivel más bajo (nivel 0) corresponde a un protocolo de acceso al medio de transmisión compartido y su función es la de transmitir y recibir paquetes a/de la línea. El siguiente nivel (nivel 1) corresponde a los protocolos de transporte los cuales permiten la comunicación entre procesos. En el nivel intermedio (nivel 2) están los protocolos que implementan servicios (v, g., transferencia de archivos, interacción con terminales, etc). El nivel superior (nivel 3) corresponde a los protocolos de más alto nivel tales como: sistemas de archivos distribuídos, base de datos distribuídas.

Los protocolos definidos inicialmente para la red se ilustran en la figura 3.

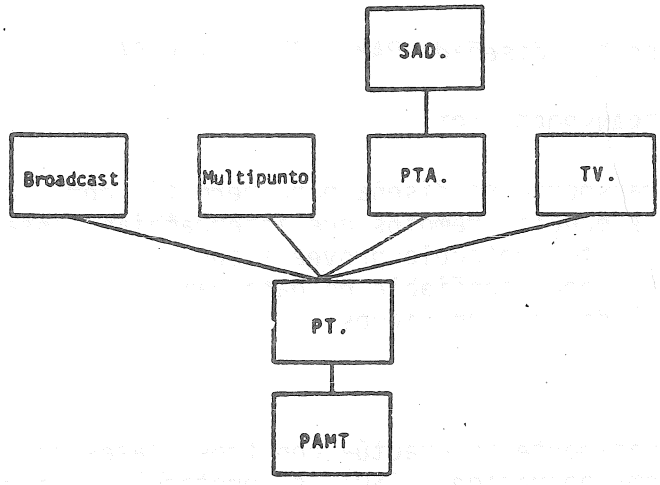


FIGURA 3 - Arquitectura Red UNIANDES

Sus funciones brevemente son las siguientes:

Protocolo de Acceso al Medio de Transmisión compartido (PAMT): Este protocolo solo entiende paquetes y corresponde a un protocolo de contención el cuaL se encarga de transmitir y retransmitir paquetes a la línea en el caso que exista alguna colisión. Este protocolo se implementa en la interfase.

Protocolo de Transporte (PT): Este protocolo solo "entiende" cartas o mensajes y dentro de él se definen dos niveles de confiabilidad: uno bastante confiable, para intercambiar cartas de una manera segura entre procesos, es decir detectar duplicados, preservar orden, controlar flujo, etc. y otro menos confiable, que ofrece un servicio de datagrama.

Protocolo de Broadcast: Es un protocolo que asegura a los procesos usuarios que un mensaje que se envíe usando esta opción, llegará a todos los nodos de la red con el nivel de confiabilidad con que se envió.

Protocolo de Multipunto: Asegura a los procesos usuarios que un mensaje que se envíe usando esta opción, llegará a un grupo de usuarios previamente definido.

Protocolo de Transferencia de Archivos (PTA): Es usado para transferir archivos entre los nodos de la red. Este protocolo resuelve incompatibilidades en la representación de los archivos, permite hacer puntos de chequeo, etc.

Protocolo de Terminal Virtual (TV): Este protocolo permite interactuar con terminales localizadas en los diferentes nodos. Resuelve incompatibilidades en cuanto a la representación de las terminales.

Sistema de Archivos Distribuído (SAD): Es un sistema diseñado para dar soporte a aplicaciones de propósito general, dando primitivas bien definidas para el diseño de aplicaciones.

En la actualidad se han diseñado PAMT, PT, PTA y SAD.

3. PROTOCOLO DE TRANSPORTE (PT).-

El protocolo de transporte se diseña para permitir comunicación entre procesos, programas en ejecución remotos que intercambian mensajes o cartas a través de la red. Este protocolo provee a sus usuarios con dos niveles de servicios: un nivel poco confiable de datagrama y un nivel superconfiable que ofrece la facilidad de conexiones.

3.1 SUPOSICIONES.

El protocolo de transporte interactúa con tres clases de procesos dentro de un nodo: el sistema operacional (SO), el protocolo de acceso al medio de transmisión (PAMT) y los procesos usuarios (PU).

Los procesos usuarios son los que demandan servicios del protocolo por medio de un conjunto de primitivas, cuya función es indicar al protocolo sobre los datos que se desean transmitir en una conexión. Estos datos son fragmentados en cartas (si se necesita) que son las que entiende el protocolo a este nivel; posteriormente las cartas son partidas en paquetes para que puedan ser manejadas por el protocolo de acceso al medio de transmisión.

A partir de este momento nos referimos al nivel superconfiable como el protocolo de transporte puesto que este nivel es más interesante dado que permite asociar a cada usuario un canal lógico seguro, por medio del cual se pueden transmitir mensajes confiablemente, aunque en realidad solo se está compartiendo un canal físico inseguro.

3.2 FUNCIONES.

El PT debe proveer métodos que aseguren un manejo confiable y eficiente de los mensajes o cartas intercambiadas entre procesos usuarios, por esto se hace necesario desarrollar mecanismos que implanten las siguientes funciones:

a. Direccionamiento : A nivel de la red existe un conjunto de direcciones lógicas que son conocidas con el nombre de puertos y el PT debe ser capaz de asociar a cada dirección lógica (PTO) la dirección física dentro de la máquina.

b. Manejo de conexiones: Incluye la forma en que se deben abrir y cerrar conexiones, de manera tal que no se permitan errores como aceptar paquetes de una conexión previamente cerrada.

c. Manejo de paquetes: Incluye el poder detectar y manejar situaciones en las que los paquetes se han dañado, perdido, duplicado o transpuesto en relación a la forma en que fueron enviados.

d. Fragmentación y ensamblaje : Dentro de una red se debe definir el tamaño máximo de los paquetes que viajan por las líneas; por lo tanto, si un usuario desea transmitir una carta que tenga un tamaño mayor al definido para un paquete, esta debe ser fragmentada en paquetes en el nodo emisor, y estos a su vez deben ser ensamblados por el nodo receptor.

e. Control de Flujo : Es el mecanismo usado por el PT para controlar que no se envíen más paquetes de los que se pueden recibir sobre cada conexión.

El Protocolo de transporte también debe poder formatear e interpretar la información de control que viaja con los paquetes, controlar los eventos concurrentes que provienen de los usuarios y además multiplexar, es decir que una sola copia del protocolo atienda a varios usuarios simultaneamente.

3.3 ESPECIFICACION.

La especificación que se encuentra en este numeral se refiere a los mecanismos de timers (FLET78). Por medio de la especificación se muestran los diferentes estados por los que puede pasar el protocolo al enviar o recibir un paquete sobre una conexión, lo cual ayuda a entender la secuencia de pasos que se deben seguir al implementar las reglas de timers en los módulos.

PARTE DEL EMISOR.

Las transiciones aquí mostradas dependen de eventos internos generados por el protocolo y eventos externos o de los usuarios.

Las transiciones son definidas en ambos casos, (emisor, receptor) por las reglas de timers.

Para ver la especificación global y total del protocolo y ademas los cambios de estado producidos por otros eventos ver (PAGA80)

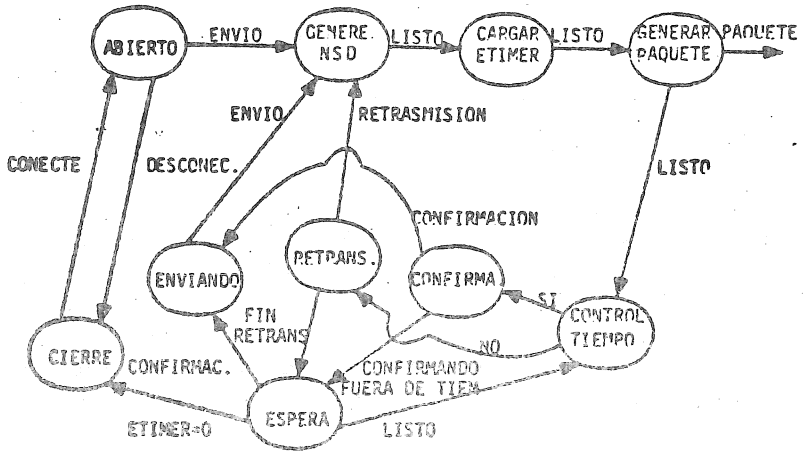


FIGURA 4a. Especificación del PT: Parte Emisor

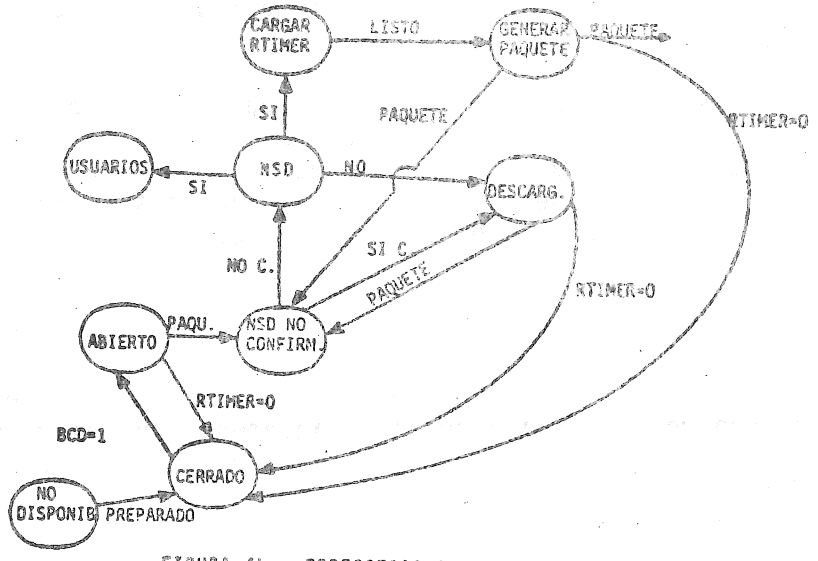


FIGURA 4b. ESPECIFICACION DEL PT: Parte Receptor

3.4 INTERFASE.-

Las interacciones que realiza el PT con sus "vecinos" (SO, PU, PAMT) son conocidas como interfases, y se hacen por medio de un conjunto de primitivas las cuales definen los eventos que se deben ejecutar.

INTERFASE USUARIO - PT:

En esta interfase se definen las primitivas por medio de las cuales se comunican el PT y el proceso usuario. Dichas primitivas son un conjunto de eventos de llamada/retorno que no se encuentran en una relación determinada.

PRIMITIVAS DE INICIACION/TERMINACION DE CONEXIONES:

- Conecte (pide una conexión entre dos puertos)
- Preparado (indica al PT de la disponibilidad de recibir mensajes de un puerto remoto).
- Desconecte (finaliza una conexión en forma diferida).
- Destruye (termina una conexión en forma inmediata).

PRIMITIVAS PARA LA TRANSFERENCIA DE DATOS:

- Envíe (indica al PT que se desea transmitir una carta sobre una conexión).
- Reciba (indica al PT que se dispone de un buffer para recibir una carta sobre una conexión).
- Cancele-envío (indica al PT que un pedido de envío ha sido cancelado).
- Cancele-reciba (indica al PT que un buffer previamente ofrecido para recibir una carta ha sido retirado).

PRIMITIVAS DE SINCRONIZACION:

- Interrumpa (utilizada para mandar mensajes prioritarios sobre una conexión).
- Reinicie (indica el reinicio normal del flujo de paquetes sobre una conexión).

En los ejemplos se dan algunas secuencias que pueden ocurrir en un normal uso del PT y además cómo es el flujo de estas a través de los módulos que implantan el protocolo.

3.5 MODULOS DEL PT.-

Los módulos funcionales del protocolo (ver figura 5) estan compuestos de un conjunto de procesos y rutinas que en unión con las estructuras de datos (tablas, colas, etc) realizan sus funciones asignadas.

A continuación se describe de una manera general cada módulo y sus funciones principales. (En los ejemplos, sección 3.6, se ilustra la descripción más detallada de un módulo.)

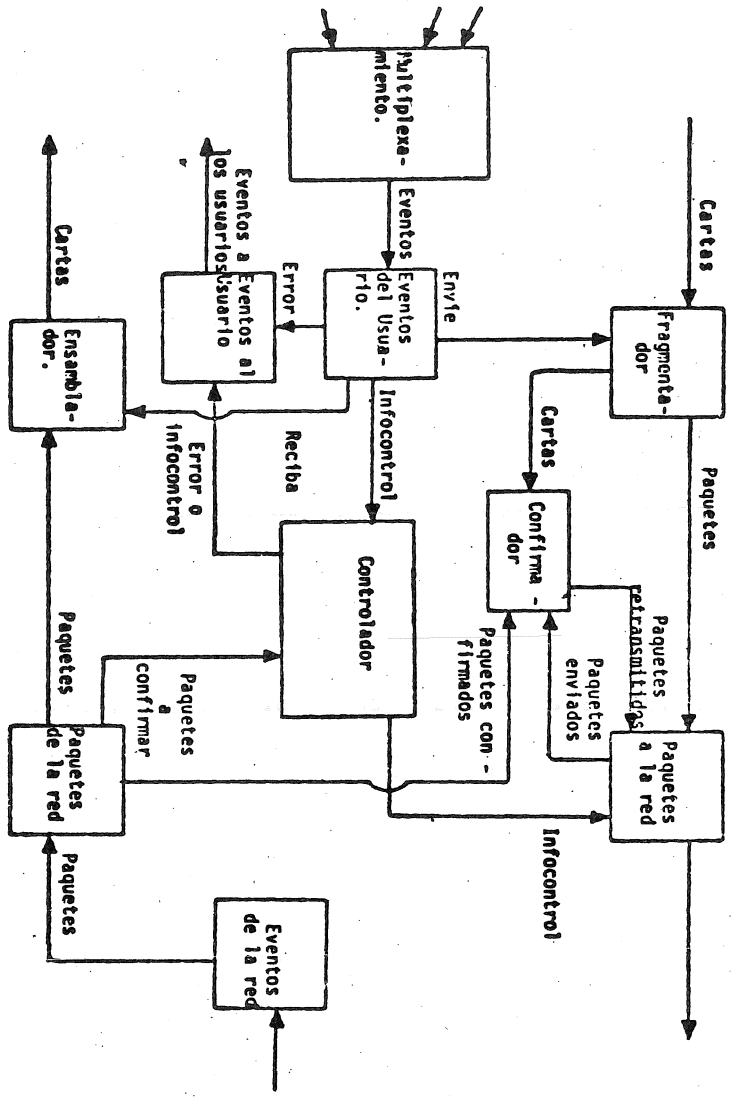


FIGURA 5. Modulos del PT

MODULO DE MULTIPLEXAMIENTO : La función de este módulo es la de controlar los eventos concurrentes producidos por los procesos usuarios del PT; esto se puede hacer por medio de alguno de los métodos de comunicación entre procesos. El módulo debe ir recolectando los eventos y colocándolos en una cola de eventos la cual será procesada por otro módulo.

MODULO DE EVENTOS DEL USUARIO: La función que realiza este módulo es la de interpretar y analizar los eventos provenientes de los usuarios (v.g: ver si los parámetros están correctos). Además se debe tomar alguna acción según el evento, por ejemplo : si es "Envíe" se debe avisar al módulo fragmentador que hay una nueva carta para enviar; esto se hace colocando la dirección y el tamaño de la carta en la cola de cartas del usuario la cual será procesada por el módulo fragmentador. Si es un "reciba" se le avisa al módulo ensamblador. Si son eventos de control (v.g. conecte, preparado,...) se comunicará al módulo controlador para que este los procese..

MODULO FRAGMENTADOR : Este módulo lo que hace es partir las cartas cuya

longitud exede a la permitida por un paquete (datos) de una manera lógica.) Crea una cola de paquetes, una por cada conexión para que el módulo que envía paquetes a la red se encargue de colocarle la información de control necesaria para ser entendido por el PT remoto.

MODULO ENSAMBLADOR: La función de este módulo es la de ensamblar cartas, para un proceso usuario, basado en los paquetes que llegan sobre esta conexión; para esto tiene una cola de paquetes recibidos de la red, la cual tiene como información la dirección de donde está el paquete y su longitud.

MODULO QUE ENVIA PAQUETES A LA RED: Este módulo es el encargado de colocar los Números de Secuencia de Datos a los paquetes y de confirmar al otro PT los datos que han arrivado, de manejar las reglas de timers del emisor cada vez que un paquete es formateado y enviado a la interfase coloca la información sobre cada uno de estos paquetes en una cola para que el módulo confirmador cada vez que reciba información sobre la entrega de cada paquete en el PT remoto lo libere.

MODULO CONFIRMADOR: Es el encargado de controlar que todos los paquetes enviados sean confirmados y cuando completan una carta, libera el buffer donde estaba la carta para que pueda ser utilizado. También tiene la función de indicar al módulo que envía paquetes a la red de una retransmisión después de pasado un intérvalo de tiempo en que un paquete que no ha sido confirmado.

MODULO DE EVENTOS DE LA RED: Tiene la función de estar pendiente de la llegada de nuevos paquetes provenientes de la interfase. Coloca la dirección donde quedó el paquete en una cola de paquetes de la red la cual será procesada por el módulo de recepción de paquetes de la red.

MODULO RECEPTOR DE PAQUETES DE LA RED: Este módulo es el encargado de procesar los paquetes provenientes de la red. Controla que los paquetes que han arrivado lleguen en el orden esperado y dentro de la ventana preestablecida. Avisa al módulo confirmador de paquetes confirmados sobre cada conexión y al módulo controlador para que confirme los elementos arrivados. También coloca la información de cada paquete aceptado en la cola de paquetes recibidos para el módulo ensamblador.

MODULO CONTROLADOR: Es el encargado de procesar los eventos de control que se ejecutan sobre cada conexión. Tiene asociada una cola de eventos de control donde va información acerca de paquetes confirmados, etc. Debe controlar situaciones como una interrupción del usuario, etc.

MODULO DE EVENTOS AL USUARIO : La función de este módulo es la de avisar a los procesos usuarios de los diferentes problemas y situaciones anormales o pedidos hechos por el usuario.

3.6 EJEMPLO;

Supongamos la siguiente conversación en dos nodos de la red:

NODO 1	NODO 2
Usuario X PT	PT Usuario Y
Conecte X, Y	Preparado X
Evento 1	Evento 1
Envíe C1	Reciba
Evento 2	Evento 2
Envíe C2	Reciba
Evento 3	Evento 3
Desconecte X,Y	
Evento 4	

En el nodo 1, inicialmente el módulo de multiplexamiento monitorea los eventos del usuario X y los pasa a una tabla de eventos, la cual es analizada por el módulo de eventos del usuario.

El primer evento es "conecte X,Y" entonces el módulo de evento mira si la conexión ya existe. Si existe, la rechaza, si no le avisa al módulo de control para que cree el registro de conexión. En este momento no se produce ningún intercambio de mensajes, sino se espera por el primer evento de envíe. Una vez el módulo de multiplexamiento haya pasado el segundo evento, (envíe C1) a la tabla de eventos el módulo de eventos del usuario se encarga de colocar la dirección y el tamaño de la carta a ser enviada (C1) en una cola de cartas del usuario.

Por medio de la cola de cartas del usuario el módulo fragmentador las convierte en paquetes, lo cual se hace de manera "lógica" puesto que en realidad solo se crea otra cola, 'de paquetes fragmentados' en la que se tiene la dirección, el tamaño de los paquetes, los números de secuencia, etc. Cada vez que una carta es totalmente fragmentada es colocada en una cola de cartas fragmentadas, para que cuando se confirme su entrega en el nodo receptor se pueda liberar el buffer donde estaba la carta.

El módulo de paquetes a la red es el encargado de procesar los paquetes fragmentados no para que estos sean enviados a la interfase una vez sean formateados, es decir después de que se coloque toda la información de control para que sea interpretada por el nodo receptor. Todo paquete que es mandado a la red es colocado en la cola de paquetes enviados no para que cuando sean confirmados por el PT remoto sean liberados por parte del módulo confirmador, el cual recibe información de control sobre paquetes que han arrivado correctamente el nodo remoto por parte del módulo de paquetes de la red.

El módulo confirmador también es el encargado de indicar al módulo de paquetes a la red cuando retransmitir un paquete que ha sido enviado y no confirmado. Por esta sucesión de pasos pasarían los eventos 2 y 3 del ejemplo.

Cuando ocurre el evento 4 "desconecte", el módulo de eventos del usuario informa al módulo de control sobre este evento y el módulo de control deja transmitir los paquetes que faltan por enviar y luego deja caer el timer del emisor a cero para asegurarse de que no quede vivo ningún paquete de esta

conexión en la red. Si mientras está en período de desconexión ocurre algún evento por parte del usuario sobre la conexión es rechazado, esto es lo que ocurre en el caso del nodo 1.

En el nodo 2 ocurriría lo siguiente: el evento 1 "preparado X" es pasado por el módulo de eventos del usuario al controlador, el cual crea un registro de conexión que se completará cuando llegue el primer paquete de uno de los usuarios especificados en la primitiva. El evento 2 "reciba" indica un buffer en el cual se colocarán los paquetes provenientes de un usuario remoto. El tamaño de los buffer disponibles determinará el control de flujo sobre la conexión. Una vez todos los paquetes han sido recibidos el controlador dejará caer el timer del receptor a cero y destruirá el registro de conexión. Los paquetes que arrivan son ensamblados en cartas por parte del módulo ensamblador.

Más detalles sobre algoritmos se encuentran en (PAGA80).

4. PROTOCOLO DE TRANSFERENCIA DE ARCHIVOS (PTA).-

Los archivos en una máquina pueden ser vistos como una colección de datos representados de acuerdo a las convenciones locales del sistema operacional (v.g; alfabeto, convención de fin archivo, etc.) Para realizar operaciones sobre estos archivos cada máquina tiene un sistema de manejo de archivos local, el cual provee funciones tales como: copiar archivos, borrar archivos, crear archivos, etc.

El problema al transferir o manipular archivos en una red, es el resolver los conflictos de las representaciones contextuales y el de extender las facilidades disponibles localmente. Para esto, es necesario diseñar un sistema (PTA) que permita estandarizar las representaciones de los archivos y proveer métodos para hacer operaciones sobre archivos a nivel de la red.

4.1 SUPOSICIONES.

El PTA interactúa dentro de un nodo con tres tipos de procesos: Proceso usuarios (PU); Sistema de manejo de archivos local (SAL); Protocolo de transporte (PT). El protocolo interactúa con los procesos usuarios por medio de un lenguaje interactivo, con el sistema de manejo de archivos locales para ejecutar las operaciones que él ofrece (copiar, borrar, etc) y con el protocolo de transporte para utilizar los servicios que él presta (v.g; establecer conexiones, recuperación y detección de pérdida y duplicado de paquetes, etc.) (Figura 6).

4.2 FUNCIONES.

COmo se dijo anteriormente el problema al hacer operaciones en los archivos de una red, es el resolver los conflictos de las representaciones contextuales y el extender las facilidades disponibles localmente. Siempre que se manipule un archivo dos partes van a tomar lugar en la operación: el usuario y el servidor. El usuario inicia las acciones y el servidor responde, es decir existe una relación de maestro/esclavo entre el usuario y el servi

dor (figura 6). Para simplificar la situación de las representaciones contextuales de un archivo, cada vez que se opere un archivo, los comandos entre el usuario y el servidor viajarán con una información de control describiendo parcial o totalmente el archivo; esta información de control puede ser generada por cualquier proceso que decida transmitir un archivo y similarmente, cualquier proceso que lo reciba puede usar esta información para completar la operación que sobre él se realice.

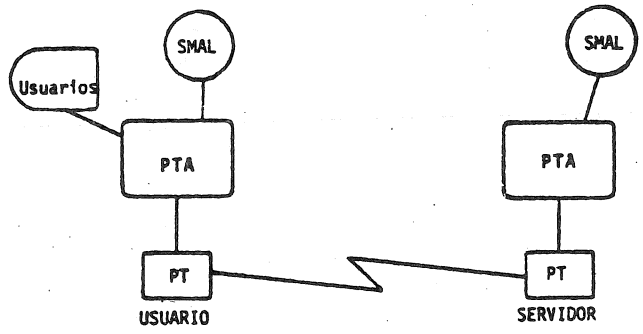


FIGURA 6 - Operación del PTA

Representación Estandar de los Archivos.

Para poder estandarizar las representaciones de un archivo en la red, se ha definido un archivo "virtual" el cual es entendido en todos los nodos de tal manera que cuando se transmita un archivo, éste debe ser convertido ("mapeado") por el PTA a la representación virtual y una vez arrive al PTA remoto será convertido ("mapeado") a las convenciones locales del nodo receptor (Figura 7).

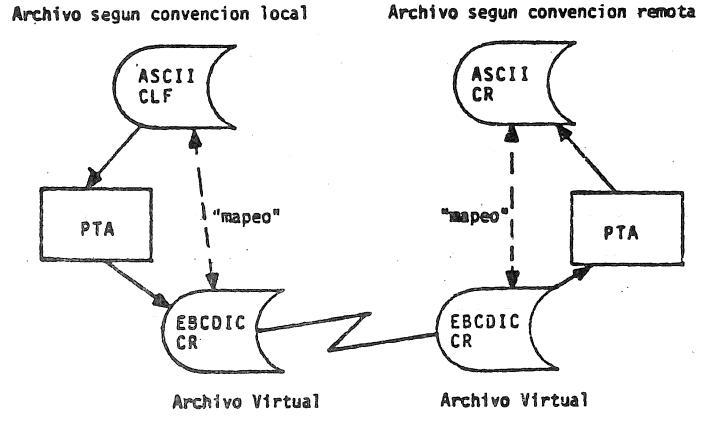


FIGURA 7 - Estandarización de los Archivos

Puntos de chequeo.

Otra de las funciones del PTA cuando está transfiriendo un archivo es el proveer una manera para realizar puntos de chequeo en la transmisión, es decir el colocar marcas de control sobre los datos del archivo enviado para que cada vez que estos arriven al nodo receptor, el PTA de este nodo confirme el recibimiento satisfactorio de ellos.

Esta función es importante puesto que si llega a ocurrir una falla en la transmisión de un archivo no es necesario retransmitirlo todo nuevamente, sino a partir del último punto de chequeo confirmado sobre los datos del archivo.

4.3 ESPECIFICACION.

La especificación descrita en esta sección muestra los estados globales del protocolo de transferencia de archivos y su interacción con el SAL y el PT.

Cada estado del protocolo es cambiado por un conjunto de eventos internos y externos, que definen los pasos a seguir por él. La especificación detallada de los mecanismos utilizados por el PTA pueden ser vistos en (PAGA80).

A continuación se describe la parte del protocolo cuando está trabajando en modo usuario y luego en modo servidor.

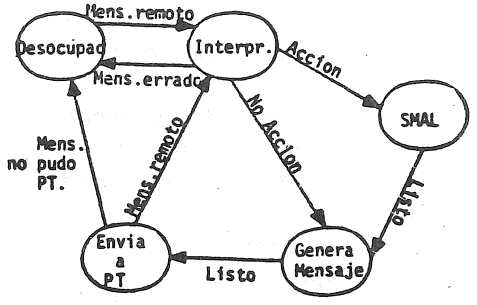


FIGURA 8a- Especificación del PTA: Modo Servidor

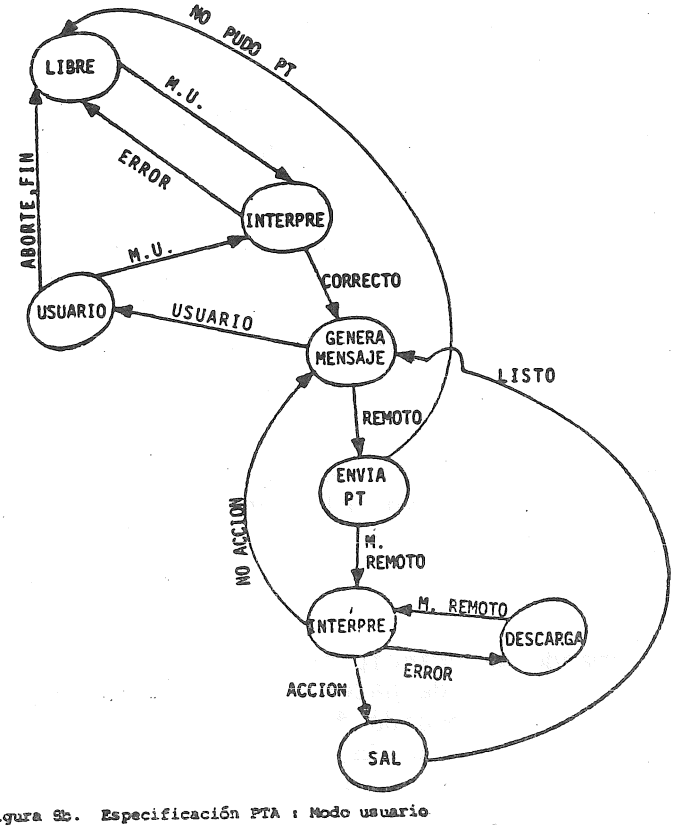


Figura 8b. Especificación PTA : Modo usuario

4.4 INTERFASE.

Las interacciones que realiza el PTA con sus vecinos (proceso usuarios, sistema de manejo de archivos local, protocolo de transporte) son conocidas como interfases y se hace por medio de un conjunto de primitivas las cuales definen los eventos que se deben ejecutar.

PRIMITIVAS DEL USUARIO AL PTA:

Son los comandos que permiten el acceso al protocolo y son utilizados por el usuario por medio de un lenguaje interactivo.

Comandos:

- Comienzo: Indica la iniciación de una serie de pedidos del usuario al protocolo.
- Tráigame: pide al servidor en envío de un archivo.
- Transfiera: Pide al servidor que almacene un archivo.
- Aborte: Anula el servicio previamente requerido.
- Cree: Demanda la creación de un archivo.
- Borre: Pide que sea borrado un archivo.
- Renombre: Cambia el nombre a un archivo.
- Copie: Hace la copia/concatenación de archivos.
- Fin: Indica el fin de los pedidos del usuario al protocolo.
- Ctransf: Implica el principio de la fase de transmisión de datos.

PRIMITIVAS DEL PTA AL USUARIO:

Estos comandos tienen por objeto informar al usuario si el servicio requerido se ejecutará o no y la culminación de él en caso de su realización.

Comandos:

- Respuesta: Informa al usuario si su pedido se puede realizar (si) si no es posible (no)
- Terminación: Informa al usuario la culminación normal o no de su pedido.

PRIMITIVAS ENTRE PTA:

Son los comandos de control de la transferencia de datos. Durante la transferencia los datos pasan desde el transmisor al receptor hasta un comando de final de datos (FD).

Comandos:

- "Archivo": Comando de principios de datos.
- "FD" : Comando de fin de datos.
- "Espere" : Ejecuta un pedido de espera (Hold (3)).
- "RR" : Pedido de restauración.
- "CPOS" : Confirmación positiva/negativa de algún evento.

En el ejemplo se dan algunas secuencias que pueden ocurrir en un normal uso de PTA y el flujo de estas a través de los módulos que implementan el protocolo.

INFORMACION DE CONTROL QUE VIAJA CON LAS PRIMITIVAS.

La información de control está compuesta por parámetros los cuales especifican un atributo y su valor. Dentro de los parámetros hay dos clases principales:

- Parámetros que sirven para identificación de un archivo (v.g.; unidad DØ1, nombre = datos, versión = Ø1, etc.)
- Parámetros que permiten negociar las características del archivo virtual (v,g; alfabeto= ascii, conversión-fín-de-línea=cr, etc).

Como la información de control es generada por cualquier proceso (usuario) que desee transferir o manipular un archivo, el PTA debe estar en capacidad de responder a todos los atributos permitidos y sus valores.

ATRIBUTOS DE LA INFORMACION DE CONTROL:

- "Unidad"
- "Nombre-Archivo"
- "Versión"
- "Nombre-Directorio"
- "Tipo" - Especifica el tipo de archivo: texto o datos.
  Datos bytes (3) de longitud 1 a n
  Texto: 8 bits (3) ASCII (3)
- "Alfabeto"
- "Convención-de-fín de línea"
- "Tamaño-bytes"
- "Marca-de-control" Especifica el número de cartas (ventana) después de las cuales se debe confirmar el recibimiento de los datos.

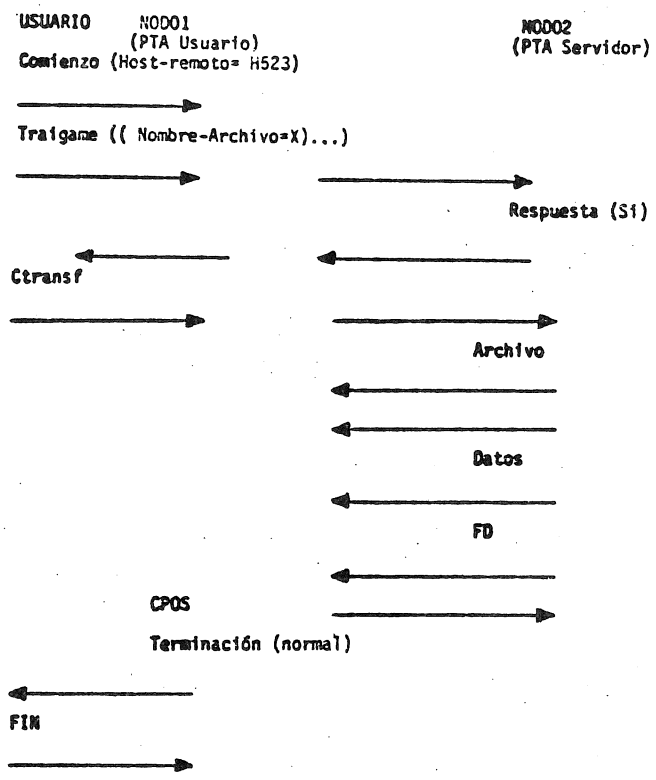
4.5 MODULOS DEL PTA.

Los módulos funcionales del protocolo (ver figura 9) están compuestos por un conjunto de procesos y rutinas que en unión con las estructuras de datos (tablas, colas, etc.) ejecutan las funciones asignadas.

La forma general de cada módulo es muy parecida a la de los módulos del PT; una descripción detallada de ellos se encuentra en (PAGA80)

4.6 EJEMPLO :

Supongamos la siguiente secuencia de eventos para la transmisión de un archivo :



El monitor de eventos del usuario se encarga de pasar los comandos del usuario al interpretador de comandos; en el ejemplo el primer comando es "comienzo", este evento luego de ser analizado sintácticamente por el interpretador es pasado al controlador, el cual crea un registro para el usuario y avisa al módulo que interactua con el PT para pedirle una conexión a un host remoto.

El segundo comando "Tráigame" pide que se envíe un archivo del nodo receptor al nodo emisor, para esto se avisa al módulo ensamblador del recurSO en el va a colocar el archivo. El controlador genera un mensaje de control para el PTA remoto indicando ese pedido; el mensaje es pasado al módulo de cartas al PT y también se crea un envíe para el PT sobre esta carta.

Después de esto se espera un mensaje de respuesta del PTA remoto el cual será pasado por el módulo de cartas del PT al controlador el que lo interpretará y por medio del módulo de comandos al usuario le avisa a este si se va a realizar su pedido.

El tercer evento del usuario "Ctransf" indica que se ejecute la acción, este evento como el anterior es pasado al controlador el cual genera un mensaje al PTA remoto. Una vez realizado esto se inicia la transferencia de datos por parte del PTA servidor, estos llegan en forma de cartas las que son chequeadas por el módulo de cartas del PT. Cada carta de las que conforman un archivo son pasadas al módulo ensamblador para que este cree el archivo en

el área de almacenamiento asignada para guardarlo. Una vez se ha terminado el proceso el controlador genera un mensaje de confirmación sobre la recepción del archivo y avisa al usuario del fin de la ejecución de su pedido, por medio del módulo de comandos al usuario.

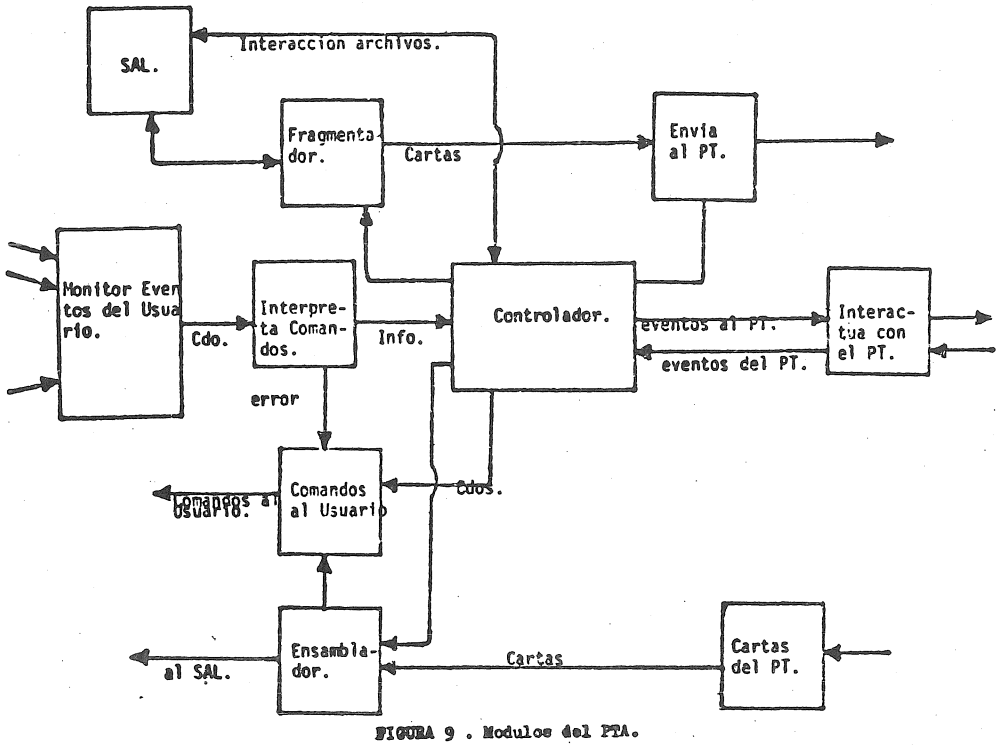


FIGURA 9 . Modulos del PTA.

5. CONCLUSIONES.

En este artículo se han descrito algunos de los factores más importantes que fueron tenidos en cuenta en el diseño de los protocolos de alto nivel (PT y PTA) para la red Uniandes.

En el desarrollo de las secciones se analizaron los aspectos más sobresalientes y su importancia dentro del diseño del protocolo. No obstante, la complejidad de dichos temas implica que sean estudiados con mayor profundidad.

El artículo no pretendía definir en su totalidad el diseño de un protocolo sino mostrar los tópicos que se deben tener en cuenta para dicho diseño.

De la arquitectura definida para red Uniandes solo se describieron el PTA y el PT porque son los protocolos básicos sobre los cuales se fundamentan los servicios y utilidades prestadas por una red de computadores. De el diseño, a la implantación de estos protocolos hay un alto grado de trabajo, que implica conocer el sistema operacional, el leguaje y el hardware de cada máqui

na.

No se debe pasar por alto decir que estos protocolos fueron diseñados para ser implantados en máquinas que permitan multiprogramación y comunicación entre procesos. Para equipos más pequeños las funciones y los mecanismos son más simples y menos complejos, por ejemplo en máquinas con monoprogramación no hya multiplexamiento, además las tablas y colas y demás estructuras de datos solo son necesarios para un solo proceso usuario.

R E F E R E N C I A S

(BOCH78) Bochmann G., Finite State Description of Communication Protocols., Danthine; Computer Network Protocols., Universite de Liege., 1978, pp F3-1-F3-10.

(BOCH80) Bochmann G., A General Transition Model for Protocols and Communication Services., IEEE: Transactions on Communications., Vol. COM 28, No. 4, Abril 1980, pp 643-650.

(BOCH80) Bochmann G., y Sunsshine C., Formal Methods in Communication Protocol Desing., IEEE: Transactions on Communications., Vol COM-28, No. 4, Abril 1980, pp 624-631.

(BOGG80) Boggs D., Shoch J. y Taft E., Pup: An Internetwork Architecture, IEEE: Transactions on Communication, Volumen COM-28, No. 4, Abril 1980, pp 612-613.

(CERF78) Cerf V., Mckenzie A., Scantlebury R., y Zimmermann H., Proposal for an Internetwoerk End-to-End transport Protocol., Computer network protocols., febrero 1978, pp H-5-H-25

(DANT80) Danthine A., Protocol Representation With Finite-State Models., IEE: Transactions on Communications., Vol. COM-28, No. 4, Abril 1980, pp 632-643.

(DOST76) Dostel J., Transmission control Protocol Specification., U.S. Departament of Commerce., Julio 1976.

(FLET78) Fletcher J., y Watson R., Mechanisms for a Reliable Timer-Based Protocol., Computer Network Protocol., febrero 1978., pp C5-1-C5-17.

(GIEN78) Gien M., A File Transfer Protocol., Danthine: Computer Network Protocol., Universite de Liege., 1978., pp D5-1-D5-7.

(HLPG77) High Level Protocol Group., A Network Independent File Transfer Protocol., Diciembre 1977.

(PAGA80) Paredes R., Gaitán L., Diseño de la arquitectura y los Protocolos de transporte y transferencia de archivos para la red local Uniandes., Universidad de los Andes., Bogotá., Diciembre 1980.

(WATS80) Watson R., Comparison Between TCP and Timer Based Protocol., ACM: Computer Protocol., 1980.

(WOOD79) Wood D., Holmgren S. y Skelton A., A cable-Bus Protocol Architecture, ACM: Sixth Data Communications Symposium, Noviembre 1979, pp 137-146.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

REFLEXIONES SOBRE PROGRAMACION CONCURRENTE Y PROGRAMACION DISTRIBUIDA

Roberto Pardo Silva

Instituto SER de Investigación
Apartado Aéreo 1978, Bogotá - Colombia.

OBJETIVO

Es bien sabido que el costo del desarrollo de software (1) continúa aumentando mientras que el de su contraparte, hardware, continúa disminuyendo. Así, la dependencia en tecnología de computadores en muchos paises se ha convertido y se convertirá aún más en una dependencia de software. Por esta razón es imperioso que en paises como los latinoamericanos se comience a dominar la complejidad del desarrollo de software, no solo cuando este se implante en arquitecturas convencionales como las centralizadas, sino cuando se desee implantar en arquitecturas sofisticadas como las distribuidas.

Este artículo se concentra en discutir conceptos fundamentales usados en lenguajes de programación concurrente e ilustrar problemas claves que tendrán que atacarse al diseñar lenguajes de programación distribuida.

TERMINOS CLAVES: Procesamiento Distribuido, Programación Concurrente, Programación Distribuida.

(1) Se usarán los términos "software" y "hardware" del inglés por no existir una traducción ampliamente aceptada de tales términos

I. INTRODUCCION

La tendencia a desarrollar sistemas de información descentralizados, es decir sistemas donde los recursos están dispersos en múltiples máquinas interconectadas entre sí, se ha ido acentuando en los últimos años. Varios factores contribuyen a este fenómeno:

1) En todas las organizaciones relativamente grandes, las decisiones basadas en información las toman diferentes personas. En muchos casos los generadores y usuarios de información están distantes. En vez de implantar un sistema de información en una sola máquina, el cual muchas veces tiende a alterar la estructura natural de la organización, es deseable implantar un sistema de información descentralizado que se adapte a la organización y permita a los usuarios tener control más directo de su información(v. g., el grupo de estadística recolecta, procesa, y analiza la información estadística sin depender de "sistemas"), delinear las responsabilidades más claramente (v. g., evita conflictos como aquellos cuando el grupo de estadística culpa al grupo de sistemas y viceversa si algo no funciona), suprimir cuellos de botellas y conflictos de prioridades (v. g., el grupo de sistemas y la máquina misma se recarga de trabajo y no es claro a qué parte de la compañía se debe dar prioridad ya que cada parte considera independientemente ser muy importante), planear más fácilmente aumentos de capacidad en el sistema(v. g., en una sóla máquina los cambios pueden afectar muchas aplicaciones mien tras que si existen varias máquinas cada una corriendo pocas aplicaciones se evitan tales efectos), etc.

2) El costo del hardware ha disminuído tremendamente. El costo de procesadores, memorias, etc, ha disminuído órdenes de magnitud hasta el punto que varias máquinas pueden ser similares en costo al de una sóla con poder computacional equivalente al agregado de varias máquinas. Además si se cuantifican las ventajas en claridad (una máquina "grande" es mucho más confusa y su software menos confiable que en las pequeñas), confiabilidad (la disponibilidad de un sistema de varias máquinas es típicamente mayor a la de un sistema con una sóla máquina), facilidad de expansión (un sistema de varias máquinas es forzosamente modular y la inversión inicial puede ser mínima), etc, el costo de un sistema de varias máquinas puede ser aún más competitivo con el de una sóla equivalente.

3) La tecnología de interconexión de máquinas, es decir, redes de computadores, ha avanzado a pasos gigantezcos en la última década. La red ARPA (DAVI79), es la pionera en la tecnología de interconexión de múltiples máquinas: se implementa el concepto de conmutación de paquetes, se desarrollan jerarquías bien definidas de protocolos, se experimentan en algoritmos de enrutamientos novedodos, etc. Los adelantos gene-

rados en gran parte por el éxito de la red ARPA son numerosos: la mayoría de los paises desarrollados están en proceso si no lo han hecho ya, de ofrecer redes públicas de datos con conmutación de paquetes; casi todos los fabricantes ofrecen arquitecturas de redes (v. g., SNA de IBM, DECNET de Digital , XODIAC de Data General, BNA de Borroughs, etc), existen experimentos exitosos del uso de redes con transmisión eficiente vía satélite y redes moviles que transmiten por ondas de radio; varias redes se han inteconectado entre sí; las redes locales han dejado su fase experimental para convertirse en productos (v. g., Z-NET de ZILOG, ETHERNET de Xerox/DEC/INTEL, RINGNET de Prime, etc). En síntesis , en la década de los 80 las arquitecturas de múltiples máquinas interconectadas entre sí serán las arquitecturas en las cuales se tendrá que implantar muchos sistemas de información.

El aspecto central de este artículo no es justificar la existencia de este tipo de arquitecturas de múltiples computadores. El punto de interés sin embargo, es tratar de reflexionar sobre las posibilidades de desarrollar buen software en tales arquitecturas. Específicamente se tratará de contestar en este artículo interrogantes como, qué conceptos de lenguajes tradicionales para arquitecturas de una sola máquina son válidos en arquitecturas de varias máquinas? Cuáles deben ser las características de un lenguaje de programación que permita expresar algoritmos para implantarse en una arquitectura de varias máquinas, es decir, algoritmos distribuidos?

Se comienza en la siguiente sección con una taxonomía de modelos de programación y consecuentemente de lenguajes de programación. En la sección 3, se menciona brevemente el modelo de programación del cual todos estamos familiarizados, es decir programación secuencial. En la sección 4 se discute en detalle los conceptos fundamentales usados en programación concurrente para contrastarlos en la siguiente sección la 5, con los problemas encontrados en programación distribuida. Finalmente en la última sección se enumeran una serie de conclusiones y se sugieren tópicos de investigación.

2. MODELOS DE PROGRAMACION

Para entender el fenómeno de programación en arquitecturas de varias máquinas y relacionarlo con la programación convencional, examinaremos brevemente en qué consiste la actividad de programar. Dado un problema se desea hallar una solución algorítmica. Típicamente el problema inicial se divide en subproblemas de menos complejidad y sucesivamente refinamos soluciones a estos problemas hasta producir un algoritmo detallado. Generalmente el final de tal refinamiento consiste en representar el algoritmo detallado en algún lenguaje de programación , es decir se produce un programa. Un programa , por lo tanto , es una de muchas representaciones a la solución de un problema.

Por otro lado, todo lenguaje de programación supone la existencia de

un ambiente de ejecución el cual es simplemente el esquema usado para interpretar las instrucciones del programa durante su ejecución. El ambiente de ejecución es importante para quien diseña e implanta el lenguaje, pero también repercute, como lo podrá apreciar el lector más adelante, en el tipo de algoritmos (y consecuentemente de problemas) que se pueden representar (resolver) .

Aclaremos lo anterior usando dos ejemplos bastante familiares. Sea el problema "buscar si un número está en una tabla ordenada" y sea el algoritmo "búsqueda binaria" . Primero pensemos en la representación de tal algoritmo en FORTRAN. El ambiente de ejecución en este caso consiste en elementos tales com el procesador de instrucciones de la máquina para la cual se compiló el programa, un espacio fijo de memoria, etc. Si por el contrario la representación del mismo algoritmo, se hace en ALGOL-60 para resolver el mismo problema, el ambiente de ejecución consiste en elementos como el procesador de instrucciones de la máquina para la cual se compiló el programa, un espacio variable de memoria, un "stack" , etc.

Aunque los dos ejemplos anteriores difieren en características importantes como la capacidad de asignación dinámica de memoria, para efectos de nuestra discusión son idénticos en cuanto a sus componentes básicos de hardware. Es decir, ambos ambientes de ejecución suponen la existencia del par procesador/memoria como los elementos primordiales para una realización física en hardware que sirva de soporte a programas en tales lenguajes.

Aclarado nuestro concepto de componentes básicos del hardware de un ambiente de ejecución, vamos a proponer en este artículo una caracterización de cuatro clases de modelos de programación, cada uno de los cuales definiendo a su vez un conjunto de lenguajes de programación (ver tabla 1).

3. PROGRAMACION SECUENCIAL

Sin duda alguna este es el modo de programación más frecuentemente usado. Se caracteriza por tener un ambiente de ejecución donde se ejecuta una instrucción a la vez; los algoritmos son independientes del tiempo; y el resultado siempre depende solo de la entrada y el programa (o sea un progrma P con unos datos de entrada E siempre producen el mismo resultado R). Los lenguajes típicos usados en programación secuencial son FORTRAN, COBOL, ALGOL-60 SNOBOL PL/I (sin subtasks), PASCAL, etc.

Muchos tipos de soluciones a problemas, sin embargo, no se pueden representar (o realizar) con este tipo de herramientas, Consideremos por ejemplo el problema " encontrar el máximo elemento de una lista de números ". Una solución del tipo "hállese simultáneamente el máximo elemento de cada mitad y luego escójase el máximo entre ellos" no se puede realizar en un lenguaje secuencial (1).

(1) Una inquietud que se deja al lector es, existen problemas no-secuenciales? o la no-secuencialidad es un artificio exclusivo de la solución a problemas?

Para	el componente básico de hardware de su ambiente de ejecución es:
PROGRAMACION SECUENCIAL	(PROC;MEMORIA)(1)
PROGRAMACION CONCURRENTE	(MULTIPLES (PROC);MEMORIA)
PROGRAMACION EN TIEMPO REAL	((MULTIPLES(PROC);MEMORIA)+RELOJ)
PROGRAMACION DISTRIBUIDA	((MULTIPLES ((PROC;MEMORIA)+RELOJ))+ SISTEMA DE COMUNICACION

TABLA 1 Modelos de Programación

(1) Un lenguaje aplicativo como LISP-puro no usaría memoria, solo un procesador en esta taxonomía.

---

4. PROGRAMACION CONCURRENTE

La programación concurrente se usa típicamente para implantar sistemas operacionales, base de datos, protocolos, etc. Además de este tipo de "programas del sistema" también se pueden usar en aplicaciones convencionales tipo "cascada" para aumentar su eficiencia. Por ejemplo consideremos una aplicación (v.g., nómina, contabilidad, control de inventarios, etc.) que se pueda abstraer en tres pasos en cascada: 1) entrada, donde se lee una transacción y se verifica su integridad, 2) procesamiento, donde se hacen cálculos y se actualizan uno o varios archivos, y 3) salida, donde se escribe alguna información del resultado de la transacción.
Si en vez de empezar a correr cada transacción únicamente al final de los pasos se tienen 3 transacciones simultáneamente cada una en un paso diferente, es posible disminuír el tiempo total de ejecución del conjunto de transacciones (v.g., si el tiempo promedio de ejecución de los pasos es comparable, la reducción total puede ser del orden de 1/3).

Este modo de programación se caracteriza por tener un ambiente de ejecución donde potencialmente se pueden ejecutar varias instrucciones a la vez; los algoritmos como tales son independientes del tiempo y el resultado, además de depender de las entradas y el pro-

grama, puede depender de otros factores (v.g., el modo de despacho de la unidades de concurrencia). Los lenguajes concurrentes típicos son PL/I (con subtasks) , ALGOL-68, CONCURRENT PASCAL, MODULA, MESA, ADA, etc.

Existen tres problemas claves que debe resolver todo lenguaje de programación concurrente:

1) El problema de manejo de la unidad de concurrencia, es decir, debe existir un mecanismos sintáctico con su semántica asociada para definir pedazos de código que puedan ejecutar concurrrentemente y mecanismos para activar y desactivar tales unidades de concurrencia.

2) El problema de sincronización, es decir, debe existir un mecanismo sintáctico con su semántica asociada para controlar el orden de ejecución de ciertos eventos y

3) El problema de comunicación, es decir, debe existir un mecanismo sintáctico con su semántica asociada para pasar información entre las unidades de concurrencia.

A continuación exploraremos el problema de sincronización por ser el más importante.

4.1 ALGUNAS SOLUCIONES AL PROBLEMA DE SINCRONIZACION

En principio existen muchos tipos de orden de eventos para controlar en un programa concurrente. Los ejemplos clásicos son ' exclusión mutua' , ' lectores y escritores con diferentes prioridades' , ' productor/consumidor' , etc. En general muchos de estos tipos de eventos suceden cuando las unidades de concurrencia comparten una estructura de datos (v.g., un archivo , un buffer). La idea clave es que un lenguaje de programación concurrente debe proveer buenas herramientas de sincronización (i.e., un modelo de sincronización), donde "buena" quiere decir una combinación de factores como generalidad , facilidad de implantación, naturalidad en su uso (v.g., de alto nivel , estructura , etc.), modularidad, y que se preste a verificaciones formales.

Examinemos entonces el papel que juegan varios modelos de sincronización en lenguajes de programación concurrente. (Las descripciones en detalle se pueden encontrar en diferentes artículos).

MODELOS DE BAJO NIVEL

Sin duda alguna el modelo más clásico es el de P/V originado por E.W. Dijkstra (DIJK68). Básicamente P y V son operaciones en objetos llamados semáforos que permiten a las unidades de concurrencia (procesos) indicar la ocurrencia de eventos . Los semáforos son objetos compartidos por las unidades de concurrencia (observación muy importante cuando se trata de extrapolar la idea de semáforos a un sistema distribuido) que tienen un valor entero y una sola cola asociada . Dado un semáforo s las operaciones definen las siguiente lógica:

```
P(s): if  s o then  s  s-1
                else  poner_en_cola
                      bloquee

V(s): s - s+1;
          active_proceso_si_existe_cola_de_espera
```

Este modelo como tal no está asociado a un lenguaje. Sin embargo, si existen primitivas en lenguajes que realizan funciones similares. Por ejemplo, los macros ENQ/DEQ sobre nombres de recursos en assembler IBM 360-370, los "statements" WAIT/COMPLETION sobre variables 'eventos' en PL/I, etc . Todas estas estructuras son equivalentes y por lo tanto poseen las mismas ventajas y desventajas .

El modelo P/V fué una gran contribución para entender y solucionar problemas de sincronización en sistemas . Sin embargo, su uso como herramienta de programación (o sea, que P y V sean las primitivas que directamente 'vea' el programador) es altamente cuestionable. Específicamente estas primitivas son de muy bajo nivel (a la assembler) y las soluciones tienden a ser obscuras (v.g., ver alguna solución de 'lectores' y escritores' con prioridad que usen semáforos), a ser poco naturales (v.g., nótese en la solución del buffer que escoger el tipo de semáforos no es trivial), a dejar esparcida la sincronización en las unidades de concurrencia en vez de asociarla en el recurso compartido que requiere la sincronización, etc.

Otro modelo muy similar, o sea con el mismo poder, ventajas y desventajas, es el de SEND/RECEIVE (BRIN70)

MODELOS DE ALTO NIVEL

El avance más fundamental en el tipo de herramientas para describir soluciones a problemas de sincronización lo constituyó la introducción de monitores (HOAR74).

Desde el punto de vista de lenguajes, un monitor es un tipo de datos con sus operaciones (procedimientos) asociadas, que normalmente abstrae el recurso compartido. Las operaciones del monitor son mutuamente excluyentes (algo que implanta el ambiente de ejecución y que es transparente para el programador).

La sincronización se realiza usando dos operaciones especiales wait y signal. La primera suspende al proceso (la unidad de concurrencia que como parte de su ambiente ejecuta una operación del monitor) y lo pone al final de una cola (nombrada como parámetro de la operación). La segunda hace que el primer proceso de la cola nombrada se active y que el proceso que invoca la operación quede en una cola de más prioridad que los suspendidos por la primera operación. Existen varias versiones del concepto de monitor. Por ejemplo, en MESA (LAMP80) solo los procedimientos ENTRY de un monitor se excluyen mutuamente y existe una primitiva llamada

broadcast para "despertar" todos los procesos de una cola. Sin embargo, conceptualmente son muy parecidos.

En general se puede afirmar que el monitor es una estructura con poder suficiente para resolver gran variedad de problemas de sincronización. Su inclusión como tipo especial con operaciones definidas por el programador es tal vez lo más significativo. Sin embargo su aspecto más débil es la necesidad de incluír un mecanismo explícito de señalización (los wait's y signal's) dentro del monitor Este mecanismo le da un "sabor a la assembler" donde no se sabe quién va a despertar a quién y le hace perder claridad a la solución. Otros problemas más detallados como el de "deadlocks" al anidar llamada a monitores y el de obligar al programador a fijar ciertas prioridades se pueden encontrar en (BLOO79).

Otro modelo de alto nivel es el de las Expresiones de Camino o "Path Expressions" (CAMP74). Básicamente este mecanismo permite especificar el tipo de sincronización de una abstracción de datos como parte de la definición misma de la abstracción. La idea es controlar el acceso a un recurso (representado en la abstracción) definiendo el orden permisible de las operaciones del recurso.
Así, existe una sintaxis especial para especificar los ordenes permisibles . Por ejemplo, el operador "+" indica selección o exclusión mutua; al operador";" indica secuencia; el operador "}{" indica concurrencia; etc.

Tal vez el aspecto más atractivo de las Expresiones de Camino es su enfoque no-procedimental, es decir, todas las restricciones de sincronización quedan capturadas en una sóla especificación donde el programador solo se preocupa del "qué debe ser la sincronización" y se olvida del "cómo" (en cambio con monitores el programador tenía que internarse en el cómo al usar wait's y signal's). Sin embargo, cuando soluciones requieren cierto tipo de prioridades o son dependientes del estado del recurso, las Expresiones de Camino deben aumentarse para poder resolver los respectivos problemas (BLOO79).

4.2 ALGUNAS LECCIONES APRENDIDAS

Resumiendo brevemente las diferentes soluciones a los problemas claves de programación concurrente se tiene lo siguiente:

a) Sincronización. Inicialmente se identificó el problema de 'exclusión mutua' y aparecieron soluciones usando LOAD/STORE (o lo que es equivalente, sin usar operadores especiales), LOCK/UNLOCK, P/V, SEND/RECEIVE, etc. Aunque estos modelos iniciales no estaban asociados con lenguajes, algunas estructuras similares (y por lo tanto muy primitivas) se infiltraron en lenguajes de alto nivel (v. g., en ALGOL, en PL/I). Luego aparecieron las regiones críticas y los monitores, los cuales representan un gran adelanto por ser buenas herra-

mientas de programación (v.g., en PASCAL CONCURRENTE, en ADA, en MESA). Varios modelos alternos se han propuesto, después del monitor: Expresiones de Camino , serializados el ACCEPT y SELECT de ADA, etc. En general un problema fundamental es el de identificar los tipos de problemas que realmente debe resolver un modelo de sincronización, ya que típicamente cada modelo es 'mejor' que otros ante ciertos tipos de problemas.

b) Unidad de Concurrencia. La construcción que normalmente se usa como unidad de concurrencia es el proceso (subtask en PL/I). Se activa mediante llamadas explícitas , ya sea mediante CALLS especiales (PL/I), INITs en lenguajes donde el proceso es otro tipo de datos (PASCAL CONCURRENTE), o llamados especiales como FORK (MESA).
Típicamente la activación produce estructuras jerárquicas y la unidad de concurrencia acaba su ejecución internamente (v.g., con un RETURN) o externamente por un ABORT de quién lo activó.

c) Comunicación. La implantación de la comunicación entre procesos se hace compartiendo memoria principal. El programador, sin embargo, realiza la comunicación a nivel de lenguaje por medio de estructuras de datos globales, o a través de los mismos mecanismos de sincronización (v.g., el monitor es un área global de comunicación entre procesos, SEND y RECEIVE además de sincronización son obviamente mecanismos de comunicación).

Se puede afirmar entonces que la tendencia primordial es diseñar herramientas estructuradas para resolver cualquiera de estos problemas . La programación concurrente lleva varios años en los cuales se han propuesto muchos tipos de soluciones, cada vez "mejores". Esta observación , desafortunadamente, es inquietante ya que si este es un dominio de problemas relativamente conocidos y sobre los cuales no existe la solución óptima , qué podremos esperar de dominios de problemas bastante desconocidos como los de los ambientes distribuidos?

5. PROGRAMACION DISTRIBUIDA

El advenimiento de sistemas distribuidos ha creado arquitecturas donde al resolver un problema, el algoritmo debe representarse en un lenguaje de tal forma que partes de código ejecutan en diferentes máquinas y a su vez estas partes interactúan entre sí. En los últimos años han aparecido gran cantidad de " algoritmos distribuidos " para resolver problemas de control de concurrencia distribuida (ELLI77, LIND79, THOM79, KANE79), de atomicidad de transacciones distribuidas (LAMP79, REED78), de deadlocks distribuidos, de procesamiento distribuido de 'queries ', etc. Sin embargo , muy poco se ha dicho de las características de un lenguaje donde se pueden representar tales soluciones. En general, la mayoría de los algoritmos se descri-

ben en lenguaje natural o se buscan formalismos gráficos (como en (ELLI77)) que están aún lejos de las descripciones "entendibles" por el sistema (1).

Volviendo a nuestro esquemas de modelos de programación se puede afirmar que la programación distribuida se caracteriza así: el ambiente de ejecución (los ambientes?) ejecutan en paralelo, o sea tiene por lo menos la complejidad de la programación concurrente; el sistema de comunicación introduce retrasos haciendo que los algoritmos dependan del tiempo, o sea tiene adicionalmente la complejidad inherente a la programación en tiempo real; y como si fuera poco muchos programas se hacen para continuar funcionando en el caso de que ciertos pedazos de código sean inaccesibles (debido a fallas de comunicación o de las máquinas mismas). Es fácil deducir que en este ambiente un resultado depende de muchos factores además de la entrada y el programa mismo.

Entre las propuestas más cercanas a lenguajes de programación distribuida están PLITS (FELD79), DISTRIBUTED PROCESSES (BRIN78), extensiones a CLU (LISK79), MOD (COOK79), etc.

Siguiendo el marco de referencia de este artículo sobre modelos de programación se plantean varios problemas claves que deben resolver los lenguajes de programación distribuida:

a) El problema de comunicación remota, es decir, deben existir mecanismos para intercambiar información entre pedazos de código distantes,

b) el problema de sincronización distribuida, es decir, deben existir mecanismos para controlar el orden de eventos que pueden suceder en máquinas distantes,

c) el problema de nombres entre nodos, es decir, deben existir mecanismos para asociar y reconocer nombres que se usen en programas remotos (v.g., nombres globales, nombres de entidades como procedimientos, procesos o módulos que sean 'exportables' a través de nodos) sin atentar contra la autonomía de un nodo,

d) el problema de confiabilidad, es decir, deben existir mecanismos para manipular estructuradamente las fallas de comunicación o de los nodos y el reinicio de éstos, y

e) el problema de tiempo, es decir, deben existir mecanismos para manipular el tiempo, el cual avanza independientemente en cada nodo.

(1) Es obvio que es útil tener formalismos de descripción 'lejanos' al sistema. Sin embargo, adicionalmente se deben tener formalismos de descripción 'cercanos' a la máquina, si nuestro interés es implantar sistemas.

El problema fundamental al resolver estos problemas (obviamente habrá otros) es el de entender qué quiere decir una buena solución , o sea una solución estructurada, flexible, que resuelva los "problemas típicos" , que se pueda implantar eficientemente, etc.

5.1 LOS PROBLEMAS DE UN AMBIENTE DISTRIBUIDO

Todo modelo de programación se basa en una serie de suposiciones con respecto al ambiente de ejecución de sus programas. Un buen entendimiento de estas suposiciones es básico para entender el tipo de problemas de interés . En particular, es interesante anotar cómo ciertos diseños en lenguajes concurrentes se han extrapolado a ambientes distribuidos, llevando implícitamente una serie de suposiciones ' fuertes ' sobre estos últimos ambientes.

Consideremos, por ejemplo, un lenguaje concurrente donde los procesos se comunican a través de mensajes . Aunque se supone que un mensaje siempre llega a su destino no se puede predecir el tiempo de retraso. Bajo este esquema (similar a lo propuesto inicialmente en (FELD79, HOAR79, BRIN78) se puede pensar que existe una gran similitud con un ambiente distribuido (v. g., no hay ambiente global y la comunicación se hace por mensajes que llegan a su destino con un retraso ilimitado pero finito.)

Esta manera de enfocar el problema (respetable por supuesto ya que ignora ciertos problemas para concentrarse en otros también importantes), desafortunadamente olvida la esencia fundamental del software distribuido, es decir, la de implantar sistemas que proveen la ilusión de ser muy confiables lo cual implica que debe atacar el problema de fallas. Dicho de otra forma , se deben diseñar herramientas que no supongan la existencia de sistemas muy confiables ya que precisamente mediante estas herramientas el diseñador desea implantar sistemas confiables. Aclaremos un poco más las características de un ambiente distribuido para entender el tipo, de suposiciones que se deben hacer en un modelo de programación distribuida.

En un ambiente distribuido una computación se modela (al igual que ambientes centralizados concurrentes) como un conjunto de procesos que cooperan. En ambientes centralizados, sin embargo, la comunicación se implanta compartiendo memoria principal. En este esquema la comunicación es casi instantánea , no se pierden mensajes y si existe una falla , casi siempre es tan catastrófica que, se aborta la computación y se comienza de nuevo.

En un ambiente distribuido los procesos son distantes. Dado que los mensajes ' viajan ' por un sistema de comunicación , existe un retraso en la comunicación. Tales retrasos tienen implicaciones en la eficiencia de la computación (v. g., construír un estado global es poco factible ya que, si se para el avance en cada nodo es demorado y se hace poco trabajo, y si no se para el avance, cuando se acaba de construír el estado ya no refleja la situación actual) y en el manejo del tiempo (v. g, cada nodo debe ser capaz de esperar por la ocurren-

cia de eventos hasta cierto límite).

Por otro lado, el sistema de comunicación de un ambiente distribuido aisla fallas, es decir, toda una máquina puede fallar sin que fallen otras. Así mismo, las computaciones se diseñan para seguir funcionando cuando ciertas partes sean inaccesibles y típicamente estas fallas se detectan mediante el manejo del tiempo (v. g., si después de cierto tiempo e intentos un nodo no se puede comunicar con otro, lo declara inoperante).

Estas características definen por lo tanto un modelo de programación distribuida que supone retrasos, existencia de fallas, e incluye la noción del tiempo.

5.2 UN EJEMPLO CONCRETO

Consideremos un ejemplo para poder analizar en concreto algunos problemas en lenguajes de programación distribuida.

El problema de interés se conoce con el nombre de "atomicidad de transacciones distribuidas ". Este se presenta en bases de datos (o inclusive en sistemas de archivos) distribuidos donde se tienen varios archivos , cada uno en un nodo o máquina diferente, y se desea que una transacción (o sea un conjunto bien definido de lectura y/o escrituras a los archivos y que tienen un sentido lógico para el usuario) actualice varios de estos archivos. La propiedad de atomicidad garantiza, que todas o ninguna de las actualizaciones de la transacción eventualmente se efectúa . Por ejemplo, sea la siguiente transacción en una aplicación bancaria que asigna a sucursales de un banco en otras ciudades, un porcentaje de un monto total de dinero para los préstamos locales. Supongamos que cada sucursal tiene sus archivos propios donde está grabada la información del monto local para préstamos.

```
Transacción préstamos (monto_total , %_A, %_B ,%_C )
        if %_A +%_B+%_C≠1 then aborte
        saque monto_total de principal;

        asigne %_A*monto_total a sucursal en ciudad A;
        asigne %_B*monto_total a sucursal en ciudad B;
        asigne %_C*monto_total a sucursal en ciudad C;

Fin transacción
```

En este ejemplo la sintaxis de nuestra transacción no es importante. Los relevante es que existe un conjunto de actualizaciones (saque' s y asigne' s) en nodos diferentes que deben realizarse completamente o no deben realizarse en lo absoluto (e. g., sería grave ' sacar' el monto total de la principal y que sólo ciertas sucursales reciban la ' asignación').

Una solución a este problema es el algoritmo del " compromiso en

dos fases" (LAM79, LIND79). A continuación describimos una versión simplificada de la solución (se sugiere al lector interesado consultar en detalle las referencias anteriores ya que existen gran cantidad de sutilezas asociadas con el algoritmo.)

La transacción se ejecuta en el nodo donde se origina (sea este el nodo coordinador). Las actualizaciones de la transacción sobre datos residentes en varios nodos, no se efectúan directamente sobre los datos sino en algún área extra (v.g., se escribe un ' track ' nuevo del disco que contiene los mismos datos del viejo pero ya actualizados). Con este mecanismo es fácil minimizar el tiempo para hacer efectivas varias actualizaciones (v.g., si solo al final se cambia un "track" que contiene apuntadores a otros "tracks", el problema de alterar varios "tracks" es equivalente al problema de alterar un solo "track"). Así, todos los nodos en los cuales hay datos que actualiza la transacción van generando suficiente información para forzar atomicidad local (aún falta la atomicidad distribuida sobre nodos).

El algoritmo realmente se invoca al final de la transacción. En este momento el nodo coordinador envía un mensaje " LISTO? " a los nodos relevantes. Si los nodos remotos han progresado hasta el punto de poder hacer su parte atómicamente , contestan con el mensaje "OK". Si todos contestan ' OK ' el coordinador puede contestarle a quien invoca la transacción que sí se harán efectivas las actualizaciones y envía un mensaje "HAGA" a todos los nodos relevantes. Si un nodo se cae (falla) antes de recibir "LISTO? " , no podrá contestar "OK" y las actualizaciones no se harán. Si el coordinador se cae antes de recibir todos los mensajes "OK" , tampoco se harán . Si un nodo se cae después de contestar "OK" y el coordinador recibe todos los "OK" de todos modos se hará la actualización ya que eventualmente cuando se reinicie el coordinador sabrá que el mensaje "LISTO? " no fué procesado. Una descripción gráfica de este proceso se ilustra en el apéndice 4.

Ahora bien, el punto para recalcar en esta discusión es que tanto la descripción anterior dada en español como la gráfica del apéndice 4, son aún bastante informales (inclusive al leer toda la descripción en (LAMP79) queda la duda si se consideran todas las posibilidades).

En la figura1 se describe el algoritmo usando un lenguaje de programación imaginario para posteriormente ilustrar problemas específicos que se presentan al programar este tipo de algoritmos . (Para efectos de esta discusión es suficiente escribir el código del coordinador).

5.3 ALGUNOS PROBLEMAS QUE ILUSTRA EL EJEMPLO

El pedazo de programa presentado en la figura . ilustra una serie de problemas de descripción en algoritmos distribuidos que son difíciles de expresar en lenguajes convencionales.

```
Coordinador: Procedure;
      /*primera fase*/
      Todobien <-True
       Do destino ={ciudad1, ciudad2, ciudad3}  while (todobien)
          send ' LISTO? '  to destino
          if respuesta ≠ ' OK '  then todobien  <- false
       end
       /*segunda fase*/
       if todobien then /*caso en que se hacen
                          las actualizaciones*/
                     Do destino = {ciudad1, ciudad2, ciudad3}
                        send ' HAGA '  to destino
                        if timeout then retransmita;
                     end
                     else/* caso en que no se hacen
                             las actualizaciones */
                     Do destino = {ciudad1, ciudad2, ciudad3}
                        send ' DESHAGA '  to destino
                        if timeout then retransmita
                     end
end Coordinador,
```

Figura 1. Parte de un Programa Distribuido

Consideremos el problema de describir el modo de comunicación. Tal como está escrito el programa la primitiva de comunicación SEND está enviando el mismo mensaje varias veces, y peor aún, se envía solo cuando se recibe respuesta del envio anterior. Idealmente sería deseable poder enviar un solo mensaje a un grupo de destinos. Sea cual fuere la semántica y sintaxis, debe haber una estrecha relación al implantar el mecanismo con los protocolos de comunicación entre procesos provistos por el software de comunicación (la eficiencia a su vez depende de la capacidad de los algoritmos de enrutamiento de la red de enviar eficientemente mensajes a múltiples destinos).

Otro problema al definir la primitiva de envio de mensajes es la definición de cuándo acaba de ejecutar. Por ejemplo un SEND podrá acabar cuando el protocolo justamente envíe el mensaje, o cuando el protocolo reciba confirmación de la recepción del mensaje (cuál sería este equivalente cuando existe un grupo distinto?), o cuando se reciba respuesta (o sea incluye procesamiento) del destino. El primer tipo de solución implica crear mecanismos en el lenguaje para recibir respuestas asincrónicamente. El tercer tipo de solución puede limitar el paralelismo. El segundo tipo es similar al primero.

Adicionalmente al problema anterior, la comunicación debe definir a dónde se envía un mensaje: a un nodo?, a un programa que puede tener varios procesos?, a un proceso? (Nótese que este problema es similar al de cómo y dónde se reciben los mensajes). Al tratar de solucionar este problema se interna el diseñador en el problema de nombres que se exportan a otros nodos. En el ejemplo se aprecia este problema: "ciudad1" es un nombre a donde se envía un mensaje y que lo conoce tanto el "coordinador" como el proceso remoto. Qué es el mínimo que se debe exportar y cómo se expresa?

Este problema de comunicación se relacionan directamente con el problema de manejo de tiempo. En el ejemplo la construcción "if timeout ..." es bastante ambigua. Primero se debe definir en qué momento se inicia el contador del tiempo y seguramente es importante que el programador tenga control sobre la longitud del intervalo de tiempo. Segundo, se debe definir si el contador va a medir la duración del tiempo para que el mensaje se transmita al otro lado o si la duración es la del tiempo para recibir una respuesta (lo cual incluye el tiempo de procesar remotamente el mensaje enviado). En cualquier caso es claro que si el contador expira, se debe procesar a nivel de lenguaje como una interrupción asincrónica (similar a la recepción de mensajes).

Finalmente, en el ejemplo no se han expresado condiciones para reiniciar el programa en el caso de fallas. Recordemos que este algoritmo es tal que si ocurre una falla durante la primera fase (v. g., se va la luz y se borra la memoria principal), al reiniciar el coordinador simplemente se olvida de lo hecho anteriormente. Sin embargo, si la falla ocurre durante la segunda fase, al reiniciar debe en principio repetir la segunda fase. Cómo expresa el programador éste tipo de lógica ? Cuál sería un "buen" mecanismo (semántica y sintaxis) para expresar este tipo de situaciones?

5.4 VISTAZO GLOBAL A ALGUNAS PROPUESTAS

Se pretende en esta sección presentar muy brevemente conceptos de algunos de los principales trabajos relacionados con programación distribuida. Es importante anotar que estos proyectos son recientes y por lo tanto en la actualidad seguramente han evolucionado mucho más de lo reportado en las referencias.

PLITS (FELD79) es un proyecto desarrollado en la Universidad de Rochester que desde el punto de vista de programación distribuída introduce varios conceptos. La noción de módulo se usa para aislar todo pedazo de código donde puede existir concurrencia, donde se pueden compartir objetos , y donde inclusive se puede programar en un lenguaje diferente al de otros módulos. Todos los módulos se comunican vía mensajes. El concepto de mensaje es introducido como un nuevo tipo de datos; cada módulo de componer, enviar, recibir, y descomponer los mensajes; y existen primitivas para manipular los mensajes (v. g., put, remove, absent, present). Los mensajes se envían a nombres de módulos y se reciben dentro de un módulo en un área común.

MOD (COOK79) es un lenguaje para programación distribuida desarrollado en la Universidad de Wisconsin . Es un descendente de MODULA y contiene las nociones de módulos y redes de módulos. Sin embargo la comunicación se hace por invocación remota y no por envío directo de mensajes. Los mecanismos de sincronización son los convencionales de sistemas concurrentes y no se propone ningún mecanismo de confiabilidad en el lenguaje mismo.

Existe una propuesta de extensión a CLU (LISK79) en la cual se intercambian mensajes explícitamente y se definen puertos para manipular a nivel de lenguaje. Sin embargo los mecanismos de confiabilidad y sincronización son los estándares para sistemas concurrentes.

Finalmente existen propuestas con modelos de computación distribuídos pero poco realistas como en (HOAR78, BRIN78)

6. CONCLUSIONES

Las reflexiones anteriores ilustran varios puntos importantes:

1) La programación concurrente es clave para aumentar la eficiencia de ciertos sistemas, mientras que la programación distribuida es necesaria para implantar algoritmos distribuidos.

2) Aunque se ha avanzado bastante en resolver problemas fundamentales en programación concurrente, estas soluciones se aplican poco a resolver los problemas fundamentales en programación distribuida (después de todo usan modelos de computación bastante diferentes).

3) Es fundamental usar "buenos" lenguajes ya que el problema no es si se puede o no implantar algoritmos concurrentes o distribuidos, sino usar herramientas que se entiendan, que sean flexibles, que sean potentes, que ayuden a disminuír la probabilidad de error y confusión del programador, etc.

4) Aunque "nuestra instalación" no posea lenguajes distribuídos (o inclusive concurrentes), estas herramientas se pueden utilizar en la actualidad como un paso adicional de desarrollo de software antes de traducir a lo convencional.

5) El area de programación distribuida está aún por explorar y lo más inquietante es que tendrá que explorarse a fondo si queremos explotar realmente las arquitecturas distribuidas que son hoy en día una realidad comercial!

REFERENCIAS

(ANDL77) Andler, S., "Synchronization Primitives and the Verification of Concurrent Programs," Carnegie-Mellon University, Mayo 1977

(BLOO79) Bloom, T., "Synchronization Mechanisms for Modular Programming Languages.", MIT/LCS/TR-211, Enero1979

(BRIN70) Brinch Hansen, P. , " The Nucleus of a Multiprogramming System" , CACM , Abril 1970, pp. 238-241

(BRIN78) Brinch Hansen, P. , " Distributed Processes: A Concurrent Programming Concept " , CACM , Nov 1978, pp. 934-941.

(BRYA78) Bryant, R. E. y Dennis, J. B., " Concurrent Programming ,MIT/LCS/TM-115, Octubre 1978

(CAMP74) Campbell, R. H. y Habermann, A. N. , " The Specification of Process Synchronization by Path Expressions, " , Lecture Notes in Computer Science 16, Springer-Verlag, 1974

(COOK79) Cook, R. P. , " MOD-A Language for Distributed Processing ", the Ist Intl. Conference on Distributed Computing Systems, IEEE, Octubre 1979, pp. 233-241

(DAVI79) Davies, D., et. al., Computer Networks and their Protocols, John Wiley & Sons, 1979

(DIJK68) Dijkstra, E. W., " Cooperating Sequential Processes" Programming Languajes (F. Genuys, Ed), Academic Press, N. Y. 1968

(ELLI77) Ellis, C. A. , "A Robust Algorithm for Updating Duplicated Databases", Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks, Mayo 1977, pp. 146-158

(FELD79) Feldman, J. A, " High level Programming for Distributed Computing " , CACM , Junio 1979 pp. 353-368

(GRAY78) Gray, J., " Notes on Data Base Operating Systems", IBM Research Lab., San José, RJ 2188(30001), Febrero 1978

(HOAR74) Hoare, C.A.R., " Monitors: An Operating Systems Structuring Concept, " , CACM Octubre 1974, pp. 549-557

(HOAR78) Hoare, C.A.R., " Communicating Sequential Processes" , CACM , Agosto 1978, pp. 666-677

(KANE79) Kaneko, A., et. al., " Logical Clock Synchronization Method for Duplicated Database Control" , The Ist Intl. Conference on Distributed Computing Systems, IEEE, Octubre 1979, pp 601-611

(LAMP79) Lampson , B. y Sturgis, H., " Crash Recovery in a Distributed Data Storage System", Xerox PARC, Marzo 1979 (aparecerá en CACM)

(LAMP80) Lampson, B.W. y Redell, D.D., " Experience with Processes and Monitors in Mesa, CACM , Febrero 1980, pp. 105-117

(LIND79) Lindsay, B.G., et. al., "Notes on Distributed Databases" , IBM Research Lab., San José RJ2571(33471), Julio 1979

(LISK79) Liskov, B., " Primitives for Distributed Computing" Proc. of 7th Symposium on Operating Systems Principles Diciembre 1979, pp33-42

(MAO80) Mao, T.W., y Yeh, R.T., " Communication Port: A Language Concept for Concurrent Programming", IEEE Transactions on Software Engineering, Marzo 1980, pp 194-204

(REED78) Reed, D. "Naming and Synchronization in a Decentralized Computer System ", MIT/LCS/TR-205, Octubre 1978

(THOM79) Thomas, R.H., " A Majority Consensus Approach to Concurrency Control ", TODS , Junio 1979, pp 180-209

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

CAPITULO F

EVALUACION, SIMULACION Y DISEÑO DE SPD

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

SIMULAÇÃO: UM MÉTODO PARA ANÁLISE DE CUSTOS E PERFOMANCE EM SISTEMAS DISTRIBUÍDOS

Jair Strack - CEEE

José Palazzo Moreira de Oliveira - UFRGS

Universidade Federal Do Rio Grande Do Sul
Pós-Graduação em Ciência da Computação
Av. Osvaldo Arnha 99, Porto Alegre, Brasil

OBJETIVO

Este trabalho visa apresentar uma metodologia experimental, baseada em modelos de simulação, para apoiar o processo de decisão na análise de alternativas para o processamento remoto interativo.

São apresentados modelos para sistemas de computação e desenvolvidas experiências de simulação visando determinar as características operacionais de sistemas configurados em estrela.

Análises de sensibilidade da solução ótima e de relação de custos entre sistemas centralizados e distribuídos são desenvolvidas a partir dos modelos.

Uma das motivações para o uso de sistemas distribuídos é a obtenção de menores custos. Esta não é sempre a única motivação, muitas vezes a escolha é decidida por um melhor serviço ou maior segurança. Entretanto o custo de um sistema é sempre importante e deve ser cuidadosamente analisado.

O custo total de operação de um sistema depende da distribuição do processamento. Aumentando o grau de distribuição alguns custos subirão enquanto outros tenderão a diminuir, é necessária uma análise destas tendências para uma decisão acertada.

A metodologia apresentada não visa a obtenção de um modelo geral que, dados certos parâmetros, forneça uma solução ideal, mas permite avaliar um certo número de alternativas com diferentes graus de distribuição.

A possibilidade de quantificar as características operacionais de diversas configurações de sistemas distribuídos fornecem, em um estágio inicial, dados para apoiar o processo decisório, diminuindo os riscos envolvidos na correta seleção das alternativas disponíveis.

A metodologia descrita decompõe o problema em duas fases distintas: a análise de desempenho das máquinas envolvidas no estudo e após considerações de custos relativos, entre uma opção centralizada e diversos casos de distribuição.

A SIMULAÇÃO COMO FERRAMENTA DE ANÁLISE DE DESEMPENHO

Foi desenvolvido um modelo de simulação, a um nível de agregação, que permite simular o desempenho de um sistema de computação atendendo a diversos usuários de "time-sharing", possibilitando determinar as possíveis relações entre as entradas e saídas, e a análise de sensibilidade quanto a variação dos parâmetros do sistema.

É empregado um modelo suficientemente geral para abranger qualquer sistema que possa ser descrito pela características acima, isto implica em resultados aproximados mas permitindo uma compreensão muito precisa sobre os mecanismos básicos de um sistema interativo, principalmente na medida em que os mesmos crescem em complexidade.

A simulação desenvolvida permite decidir quanto a exequibilidade de uma dada aplicação, tendo em vista restrições quanto ao tempo de resposta e "throughput" dos sistemas considerados, fornecendo indicações sobre as possibilidades de desempenho de diversos equipamentos submetidos a uma dada carga de trabalho.

Dentro dos objetivos propostos, a simulação é um ele

mento auxiliar para a tomada de decisões a médio para longo prazo em situações que envolvam custos e riscos elevados. Esta afirmativa deve-se, principalmente, a duas razões. A primeira é o custo que tende a ser elevado devido ao treinamento necessário do pessoal incumbido de desenvolver o estudo, à preparação dos dados de entrada, que é uma tarefa que requer tempo considerável e, finalmente, devido aos custos de processamento. A segunda prende-se ao grau de agregação que é, como já foi dito acima, bastante grande devido à utilidade geral a que se destina o estudo, além disto, uma simulação não atinge, normalmente, um alto grau de acuracidade devido a dificuldade de implementar um alto nível de detalhamento no modelo. Como decorrência destas restrições devemos limitar o escopo de nosso trabalho a escolha da configuração e do porte dos seus elementos constituintes.

O MODELO DE SIMULAÇÃO

O modelo selecionado é o apresentado graficamente na fig. 1, que é de uso geral na bibliografia e que, caso os parâmetros de entrada forem escolhidos criteriosamente pelo modelista, representa o sistema real com uma acuracidade bastante boa.

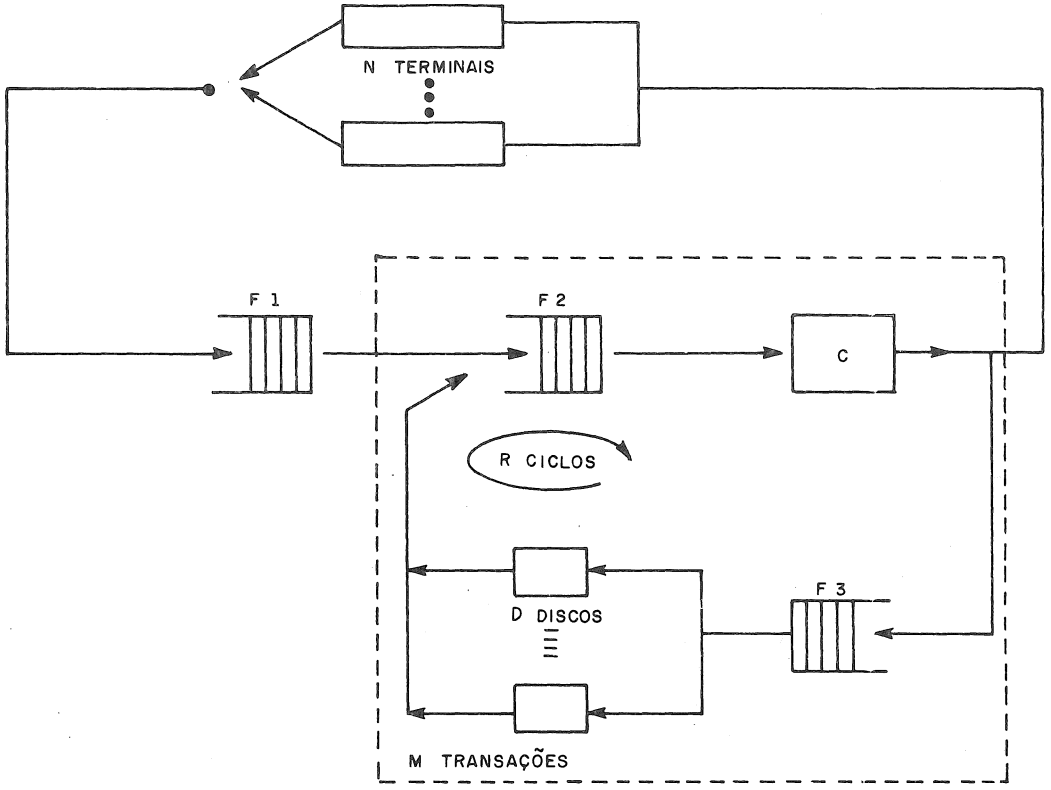
O elemento, básico, neste modelo, é a transação a qual é entendida como sendo o conjunto de operações solicitadas por um comando do usuário e satisfeira por uma mensagem do computador.

Existem, em um dado momento, N terminais ativos interagindo com o computador por meio de transações. Cada comando de usuário identifica uma determinada rotina que é carregada na memória e executada, dando lugar a R acessos a disco no decorrer do processamento da transação.

É definido um determinado nível de multiprogramação M que pode ser variado para permitir uma melhor utilização da máquina. O nível de multiprogramação, juntamente com a tecnologia empregada, causam um certo "overhead".

Finalmente existem unidades de disco com D canais independentes de acesso de uso compartilhado pelas transações ativas.

O modelo apresentado não é adequado à simulação de máquinas de pequeno porte (minicomputadores) devido a características específicas deste tipo de equipamento. A principal delas é a operação em monoprogramação e a utilização de técnica de "swap-out", "swap-in" ao passar o atendimento de um tipo de transação para outro. Em decorrência da monoprogramação não existem as fitas associadas ao processador e ao disco. Aplicadas estas características ao modelo da fig. 1 é obtida uma representação adequada aos minicomputadores que serão utilizados



Modelo de uma Máquina Multiprogramada (de grande porte)

fig. 1

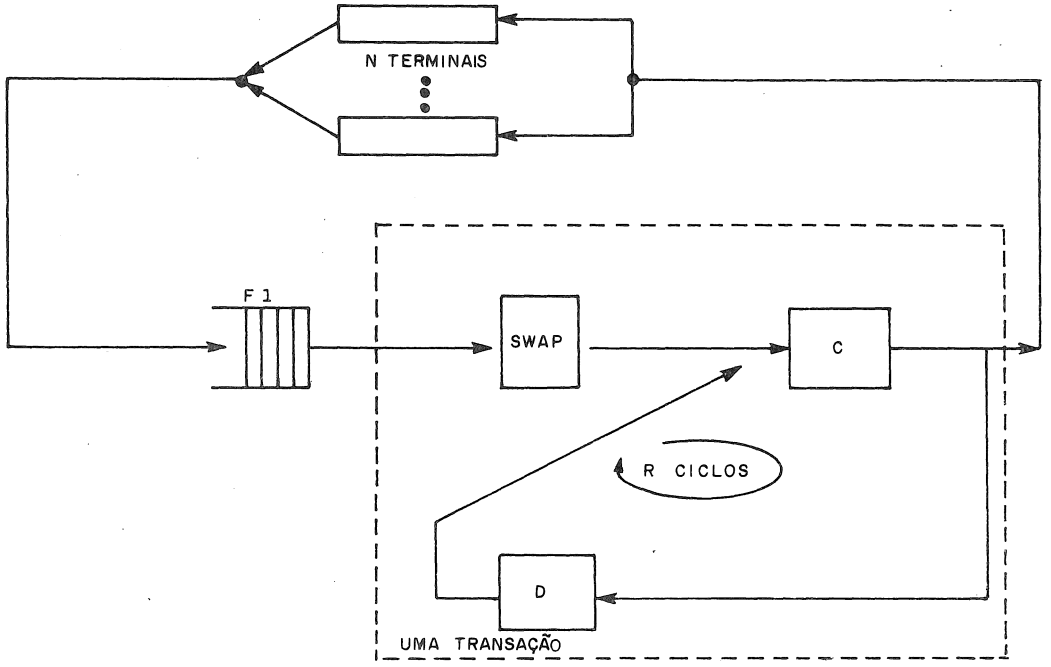


fig. 2 - Modelo de um Minicomputador Monoprogramado

como processadores remotos nos sistemas distribuídos (fig. 2).

ANÁLISE DE PERFORMANCE DE UMA MÁQUINA MULTIPROGRAMADA (GRANDE)

Os parâmetros para a experiência de simulação, com o modelo da fig. 1, foram definidos de forma a representar uma máquina de porte razoavelmente grande, sem que fosse selecionado um equipamento específico mas procurando ser representativos desta classe de computadores.

PARÂMETROS:

Velocidade da CPU = 1.000.000 instruções/seg
Nível de multiprogramação = 3
Número de canais de I/O = 2
Tamanho de uma transação = 100.000 instruções
"Overhead" por transação = 0,08 seg
Número de I/Os por transação = 10
Tempo por I/O = 0,03 seg
Tempo de raciocínio do usuário = 15 seg.

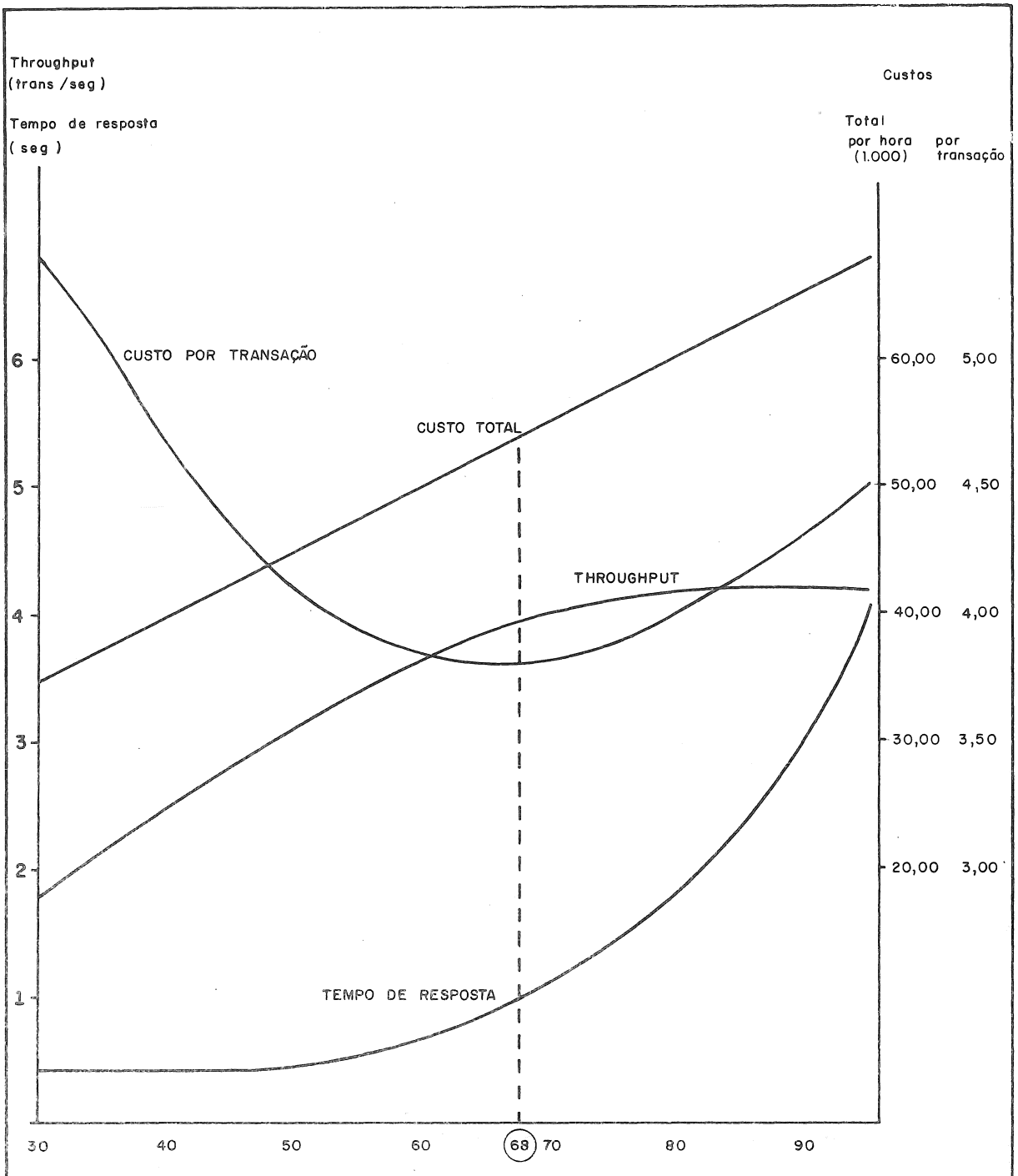
O modelo de simulação foi codificado na linguagem GPSS gerando um programa que executado, para diversas configurações de terminais ativos, permite obter os valores de tempo de resposta e "throughput", medido em transações por segundo. Uma execução típica, em IBM/370 modelo 145, com um tempo real de 20 minutos, sendo 10 minutos de inicialização e 10 minutos de simulação com obtenção de estatísticas, utiliza cerca de 5 minutos de CPU.

A seguir é calculado o custo de operação do computador, pela soma de duas parcelas correspondentes ao custo da máquina central e ao custo dos terminais ativos, estes valores são globais incorporando "hardware", instalações e pessoal. Os custos considerados foram:

Computador = Cr$ 20.000,00/hora
Terminal = Cr$ 500,00/hora

Dividindo-se o custo total, por segundo, pelo "throughput" obtém-se o custo unitário por transação. Estes valores encontram-se representados graficamente na fig. 3 onde pode-se notar um nítido mínimo, no valor do custo unitário. Selecionando-se este ponto é obtido o tempo de resposta e a capacidade de processamento do sistema para o custo mínimo. Estes são os elementos básicos para a decisão sobre a exequibilidade do sistema em análise.

Caso os resultados da simulação não se enquadrem nas limitações do projeto devem ser modificados os parâmetros do sistema até ser atingida uma configuração adequada que, uma vez obtida, fornecerá o custo por transação que será utilizado na avaliação de desempenho e custo relativo entre a configuração centralizada e as diversas configurações distribuídos.



Resultados da Simulação de uma Máquina Multiprogramada (grande)

fig. 3

O mesmo modelo permite desenvolver uma série de análises considerando a variação dos diversos parâmetros utilizados. Na figura 4, foi considerada a configuração ideal de 68 terminais ativos e a variação dos parâmetros: ciclos de I/O, instruções por ciclo e tempo de acesso a disco.

Com esta análise é possível determinar a sensibilidade da solução ótima a eventuais modificações que devem ser introduzidas nas transações ou a alterações na localização física dos arquivos nas unidades de disco alterando o tempo de acesso. Os resultados são apresentados graficamente na fig. 4.

ANÁLISE DE PERFORMANCE E CUSTO COMPARADA ENTRE SISTEMA CENTRALIZADO E DISTRIBUÍDOS.

Para a realização das experiências de simulação com o modelo da fig. 2 foram especificados os seguintes parâmetros:

Velocidade da CPU = 100.000 instruções/seg
Tamanho de uma transação = 100.000 instruções
Tempo de "swap" 2 0,3 seg
"Overhead" por transação = 0,2 seg
Número de I/Os por transação = 10
Tempo de I/O = 0,054 seg
Custo do computador = Cr$ 3.000,00/hora
Custo do terminal = Cr$ 350,00/hora

Nos casos de menor custo, por transação, para os modelos multi e monoprogramado foram obtidos os valores a seguir:

	Multiprogramada (grande)	Monoprogramada (mini)
Throughput	3,95 tr/seg	0,51 tr/seg
Tempo de resposta	1 seg	2,6 seg
Custo total	Cr$ 48.000,00/hora	Cr$ 6.150,00/hora
Custo transação	Cr$ 3,80	Cr$ 3,34
Número terminais	68	9

A partir destes dados pode ser realizada a comparação entre um sistema centralizado, atendendo uma rede de terminais em estrela, e um totalmente distribuído.

No totalmente distribuído, constituído por uma rede de minicomputadores é suposto que todos os dados necessários ao processamento de uma mesma transação estejam armazenados em apenas um nodo. O sistema de aplicação global pode ser decomposto em áreas geográficas bem definidas, de tal modo que cada área é atendida por um minicomputador.

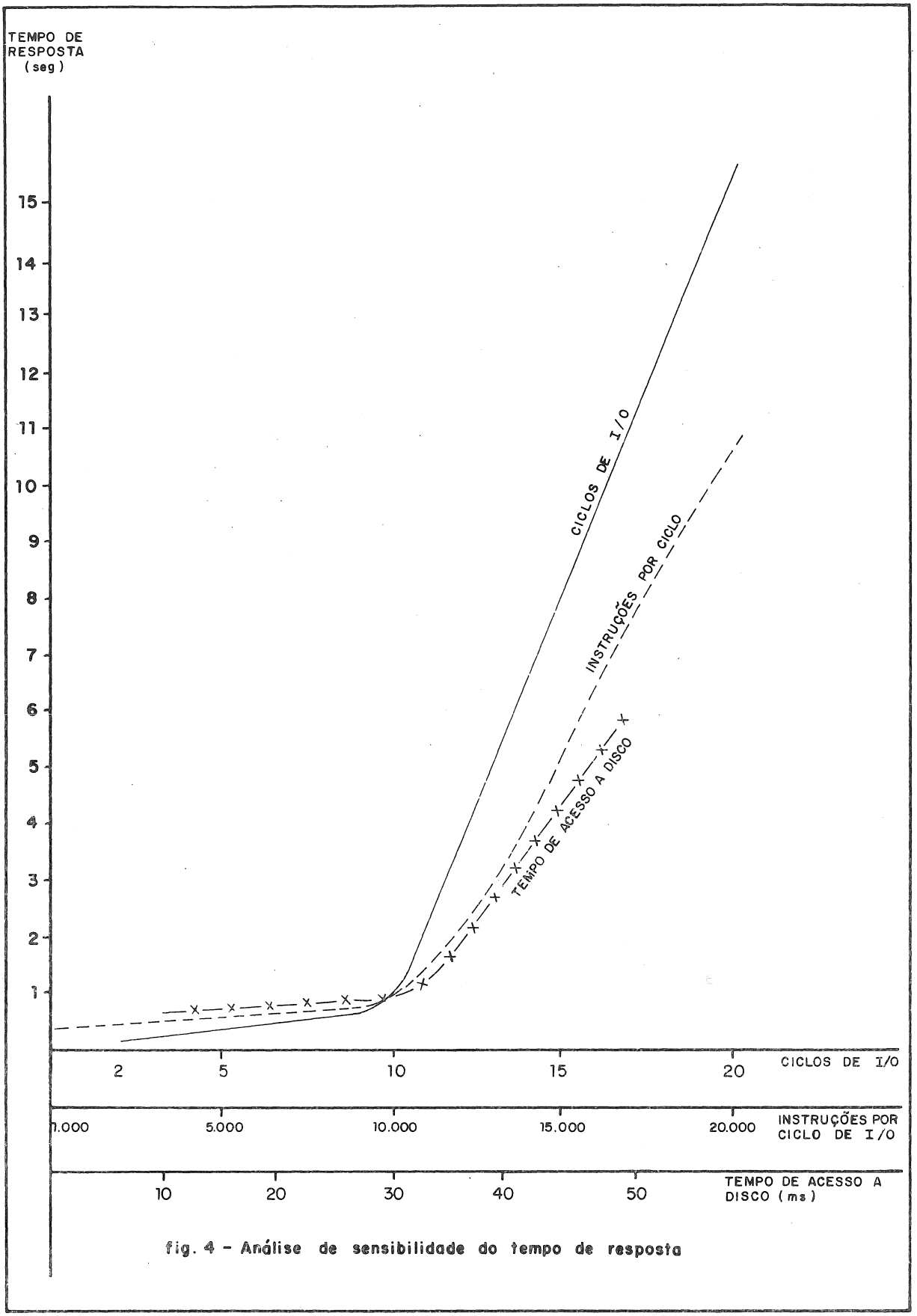


fig. 4 - Análise de sensibilidade do tempo de resposta

Caso uma transação necessite dados que não estejam no mesmo nodo ao qual está ligado o terminal é necessário que este nodo contenha informações suficientes para determinar aonde estão armazenados os dados para o processamento. O custo de transmissão de dados, por transação, é considerado igual ao do sistema centralizado, pois quanto menor o número de transações remotas, menor poderá ser a velocidade dos canais de telecomunicação empregados pelo sistema. Finalmente o custo de processamento para uma transação remota será igual a duas vezes o do processamento de uma transação local e é dividido igualmente entre os dois nodos envolvidos no processamento.

Dadas as características acima o custo, por transação, associado ao sistema centralizado será:

$$C_{centr} = C_g + C_t$$

onde C_g = custo por transação obtido a partir da simulação de uma máquina de grande porte.

C_t = custo de transmissão de dados por transação

e o custo para o sistema distribuído será:

$$C_{distr} = PC_m + (1 - P)(2C_m + C_t)$$

onde C_m = custo por transação obtido a partir da simulação de um minicomputador.

Para determina o ponto a partir do qual será mais econômico empregar um sistema distribuído, é calculada a relação entre o custo C_{distr} e o custo C_{centr}.

A reta definida pela relação de custos igual a um separa as zonas onde é economicamente aconselhável o processamento totalmente distribuído e aquela onde é mais adequado o centralizado (fig. 5).

Outra configuração de sistema distribuído a ser considerado é o sistema hierárquico composto por um nodo central, constituído por uma máquina de grande porte, ao qual estão conectados diversos minicomputadores.

Inicialmente é considerado que o nodo central contenha uma cópia total da base de dados e que os nodos periféricos, contenham o subconjunto da base global referente a sua região geográfica.

Uma transação pode ser processada localmente ou, caso os dados não estejam disponíveis em seu nodo, deve acessar o nodo central onde encontrará os dados necessários. O custo de transmissão, por transação, é considerado igual ao custo no ca

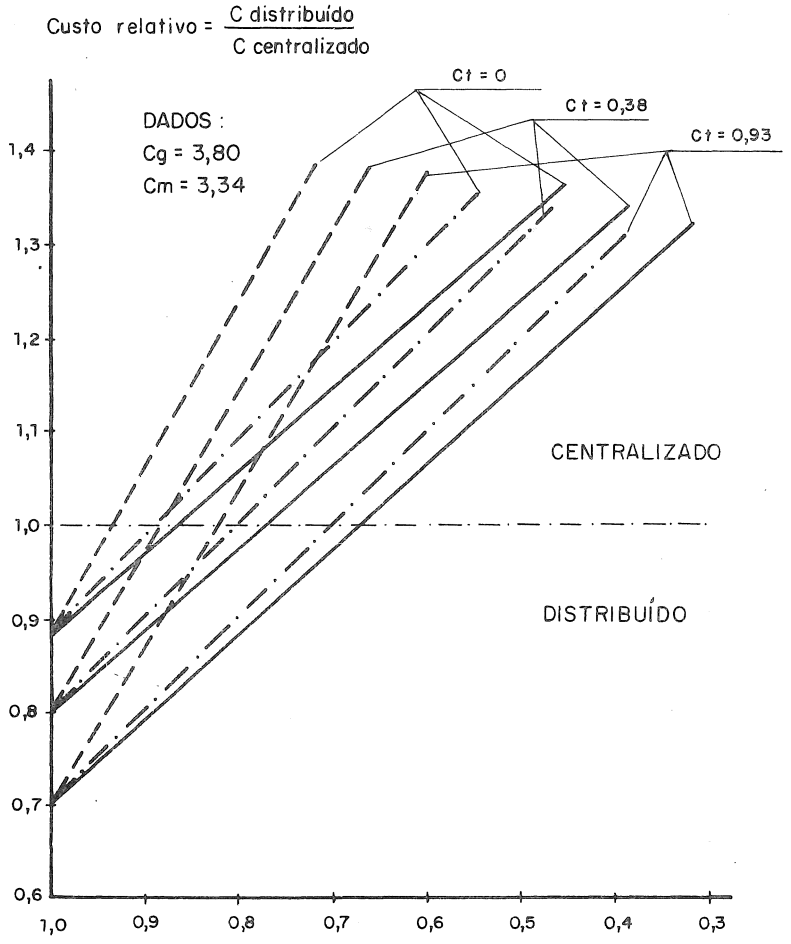


fig. 5 – Sistema totalmente distribuído ————

Sistema hierárquico { **Nodo central sem cópia da base de dados** — — —
Nodo central com cópia da base de dados —·—·—

so centralizado.

A relação entre os custos será:

$$\text{Custo relativo} = \frac{C_m}{C_g + C_t} + (1 - P),$$

os resultados da avaliação desta expressão estão representados graficamente na fig. 5.

Uma outra forma de operação, de um sistema hierárquico, é desenvolvida para evitar a necessidade de manutenção de uma cópia central atualizada de toda a base de dados. Este custo pode ser consideravelmente alto caso a taxa de alterações for elevada. Neste modo de operação caso uma transação não possa ser atendida localmente, deverá acessar o nodo central de onde será dirigida para o nodo que contém os dados necessários ao processamento. O custo do processamento da transação no sistema distribuído será:

$$C_{distr} = PC_m + (1 - P)(2C_m + C_g + 2C_t)$$

e a relação entre os custos:

$$\text{Custo Relativo} = \frac{PC_m + (I - P)(2C_m + C_g + 2C_t)}{C_g + C_t}$$

cuja avaliação encontra-se representada na fig. 5.

CONCLUSÕES

O método apresentado permite uma avaliação prévia de configurações de sistemas de informação distribuídos, utilizando, para predizer o desempenho de cada nodo, modelos de simulação. Estes modelos permitem obter as mais importantes características operacionais das máquinas utilizadas. Constata-se que o emprego de modelagem requer uma compreensão bastante clara dos mecanismos de funcionamento dos equipamentos, bem como um estudo criterioso das características das transações envolvidas e do sistema de aplicação. A utilização de variantes do modelo básico permite representar outras categorias de sistemas operacionais e configurações de "hardware".

A análise, propriamente dita, dos sistemas de informação distribuídos utiliza os dados obtidos por meio da simulação para avaliar analiticamente diversas configurações.

Finalmente, deve ser salientado, que o exemplo estudado representa unicamente um caso específico. A metodologia apresentada pode ser facilmente adaptada para um sistema deter-

minado, permitindo uma análise de custos e desempenho em uma fase inicial de projeto. Desta forma podem ser selecionadas as melhores alternativas para um estudo mais detalhado, minimizando os custos e riscos envolvidos no mesmo.

BIBLIOGRAFIA

OLIVEIRA, José Palazzo M. Minicomputadores um caminho na distribuição do processamento. In: CONGRESSO REGIONAL DE MINICOMPUTADORES, 1, Recife, maio 1979 Anais SUCESU, 1979. p. 9-15.

OLIVEIRA, José Palazzo M. Processamento distribuído-Características de uma nova metodologia. In: XII CONGRESSO NACIONAL DE PROCESSAMENTO DE DADOS, São Paulo, outubro 1979. Anais SUCESU, 1979. p. 255-257.

BOYSE, J.W. & WARN, D.R. A straightforward model for computer performance prediction. Computing Surveys, 7 (2): 73-93, June, 1975.

DENNING, Peter J & BUZEN, Jeffrey P. The operational analysis of queueing network models. Computing Surveys, 10 (3):225-261, Sept., 1978.

BUCCI, Giacomo & STREETER, Donald N. A methodology for the design os distributed information systems Communications of ACM, 22(4):233-245, Apr. 1979.

SIMULATION: its place in performance analysis. EDP Performance Review, 1(11):1-5, Nov., 1973.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

SSIP: SIMULADOR DE INTERCONEXÃO DE PROCESSADORES

M. Tazza

R. Weber

Ph. Navaux

Universidade Federal do Rio Grande do Sul
Pós-Graduaçao em Ciência da Computaçâo
Av. Osvaldo Aranha 99, Porto Alegre, Brasil

RESUMO

O estudo de sistemas a múltiplos processadores desenvolveu-se muito estes últimos anos com o advento dos microprocessadores. Por conseqüência, tornou-se necessário criar ferramentas capazes de melhor determinar as características e desempenhos necessários para estes sistemas.

Este artigo trata do desenvolvimento de uma destas ferramentas, um simulador de interconexão SSIP, que tornou-se necessário para o projeto de um sistema multimicroprocessador SMM em execução na Universidade Federal do Rio Grande do Sul (UFRGS). Este simulador permite testar diferentes interconexões entre processadores e memórias, de maneira a verificar seu desempenho relativo e portanto auxiliar na definição de uma arquitetura melhor adaptada ao SMM.

É apresentado o núcleo do SSIP que simula a interconexão entre os processadores e memórias de um dado sistema multiprocessador.

O SSIP foi definido buscando flexibilidade para permitir que a disciplina básica de estabelecer ligações entre processadores e memórias pudesse ser alterada facilmente sem necessidade de mudanças na estrutura do núcleo.

Este trabalho é parcialmente financiado pela FINEP - Financiadora de Estudos e Projetos.

OBJETIVO

O objetivo deste artigo é apresentar os resultados obtidos no desenvolvimento de um Sistema Simulador de Interconexão entre Processadores e Memórias (SSIP). Descreve-se o sistema simulado e a estrutura lógica do simulador, apresentando-se ao final futuros desenvolvimentos. A utilidade deste simulador é permitir o estudo comparativo de diversas estruturas de interconexão medindo-se seus desempenhos relativos.

I. INTRODUÇÃO

O advento do microprocessador abriu novas portas para o arquiteto de sistemas, permitindo obter sistemas mais potentes, melhor dirigidos para as aplicações, com melhor segurança e a um baixo custo. Os primeiros projetos desenvolvidos com microprocessadores visavam usá-los para substituir o "hardware" de controladores. Hoje, no entanto, de mais em mais desenvolvem-se projetos com múltiplos microprocessadores; estes sistemas permitem obter uma maior potência de saída devido ao processamento paralelo, bem como maior tolerância a falhas.

Dois tipos principais de sistemas decorrem do uso de múltiplos processadores: os sistemas de microprocessadores distribuídos e os sistemas multimicroprocessadores. No primeiro caso o processamento é distribuído entre processadores que estão fisicamente interconectados mas que possuem um grau de acoplamento lógico variável (Thu.78). Por outro lado um sistema multiprocessador é um sistema com processadores semelhantes que partilham o acesso à memória comum, canais de entrada e saída, unidades de controle e periféricos, com um único sistema operacional controlando-o (Ens.75). O trabalho apresentado neste artigo concerne este último ítem.

Um dos aspectos mais importantes no estudo dos sistemas com múltiplos processadores é a interconexão que permite a comunicação entre estes. Como conseqüência, diversos estudos e classificações (And.75) apareceram nos últimos anos tentando estabelecer o melhor esquema de interconexão para determinados sistemas (TRI.79).

No desenvolvimento do projeto de um Sistema Multimicroprocessador (SMM) na UFRGS foram sentidos também os problemas inerentes à escolha da interconexão melhor adaptada à comunicação entre os processadores e as memórias. Tornou-se portanto necessário simular primeiramente o multimicroprocessador e, especificamente, a interconexão, para permitir uma escolha da arquitetura final com uma melhor base de conhecimentos dos desempenhos. No estudo da arquitetura do sistema precisou-se de uma ferramenta de software que permitisse especificar uma determinada arquitetura e acompanhar certos aspectos do seu comportamento. Para obter esta ferramenta, era viável desenvolver um simulador baseado numa Linguagem de Descrição de Hardware (LDH) que fizesse o mapeamento entre a arquitetura descrita e uma representação

interna capaz de ser simulada e avaliada quanto ao seu desempenho.

Dentro das LDH, diversos níveis de descrição são possíveis, como pode ser visto na tabela 1. O nível da LDH aconselhada para o estudo do SMM seria, pela classificação de Barbacci, o nível 1. Este aborda os aspectos de funcionamento e comunicação entre processadores, memórias, chaves e periféricos e portanto adapta-se perfeitamente ao estudo necessário sobre a interconexão do SMM.

Nível	Elementos
1. De sistema	processadores, memórias, chaves, periféricos
2. De programação	instruções de máquina e operação
3. Transferência de registradores	registradores, transferências entre registradores
4. Circuitos de chaveamento	subníveis seqüencial e combinacional
5. Circuitos	diodos, transistores, resistores

Tabela 1 - Classificação de Barbacci

No entanto, decidiu-se pela implementação, primeiro, de um núcleo do sistema simulador que permitisse melhor avaliar as necessidades da LDH. Esta decisão foi conseqüência do objetivo de se ter, a curto prazo, um sistema para definir diferentes ligações entre processadores e memórias e que possibilitasse acompanhar seu comportamento quando submetido a pedidos de processadores solicitando um acesso a memórias através do sistema de interconexão. Este núcleo forneceria, posteriormente, os elementos para a definição de uma LDH que possibilitasse a simulação de todo o SMM.

II. O SISTEMA A SER SIMULADO

O sistema a ser simulado pode ser visto, de forma genérica, na figura 1.

Na figura 1, P1, P2,... Pn representam processadores e M1, M2,...Mm representam módulos de memória. Os processadores e as memórias estão interligados através de uma estrutura de interconexão, que pode caracterizar diversas arquiteturas multiprocessadores. Por exemplo, a interconexão pode ser um barramento único, que interligue todos os elementos e seja utilizado de forma

multiplexada no tempo. Outra forma de estruturar a interconexão é através de uma rede crossbar (figura 2).

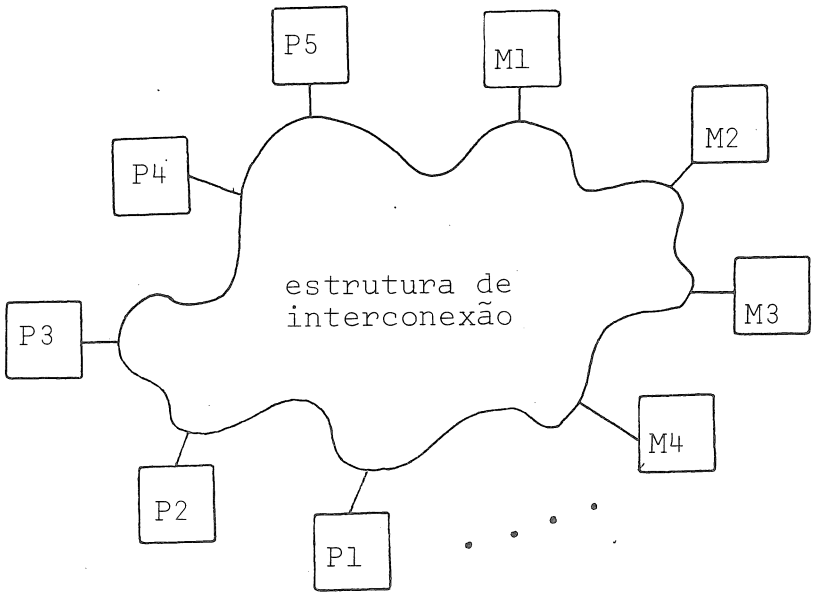


Figura 1 - Sistema multiprocessador

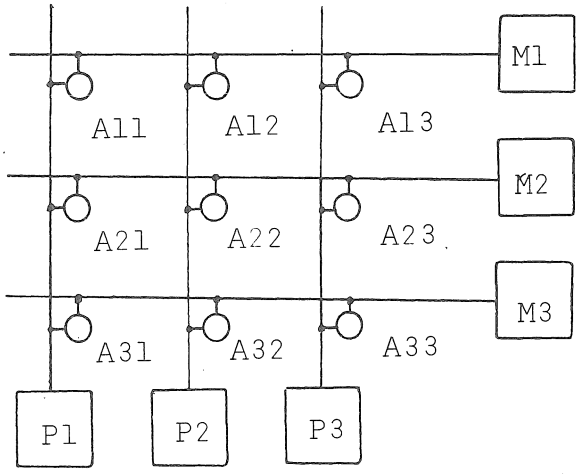


Figura 2 - Sistema com rede crossbar

A comunicação entre um determinado processador e uma memória é feita interligando-se os seus dois barramentos através de chaves. Por exemplo, se na estrutura da figura 3 o processador P2 deseja estabelecer contacto com a memória M1, isto é feito através da chave S21. Note-se que, estando esta ligação estabelecida, tanto o barramento que parte do processador P2 como o da memória M1 não podem mais ser utilizados por outros elementos.

Diversas outras estruturas de interconexão podem ser definidas, caracterizando assim diversos sistemas multiprocessado-

res (tri.79).

O Simulador reproduz o processo de estabelecer as ligações entre processadores e memórias para a estrutura de interconexão de uma determinada arquitetura multiprocessador. Desta maneira, o Sistema Simulador substitui a estrutura de interconexão mostrada na figura 1.

Para reproduzir o comportamento da interconexão quando solicitada para estabelecer o caminho, o Sistema Simulador necessita de vários elementos:

- Descrição interna da interconexão.
- Representação interna do estado de utilização da interconexão.
- Um conjunto de rotinas capazes de realizar análises e decisões quanto ao estabelecimento de um caminho de comunicação: os arbitradores.
- Um controle que reconheça os pedidos dos processadores, efetue as chamadas dos arbitradores para estabelecer a conexão desejada e simule o uso desta conexão.

Na próxima seção, descreve-se a estrutura global do sistema simulador. A notação utilizada é uma modificação de "funciogramas" (Ric.77). Um funciograma é um grafo com três tipos de nodos: ESTAÇÕES, ÁREAS e CANAIS DE AVISOS (figura 3).

Figura 3 - Elementos de um funciograma

As estações são elementos ativos e representam processamento. Podem ler/escrever em áreas de dados. Estas operações são representadas por arestas dirigidas da área para a estação (leitura) ou da estação para a área (escrita). A aresta dirigida nos dois sentidos indica leitura e escrita.

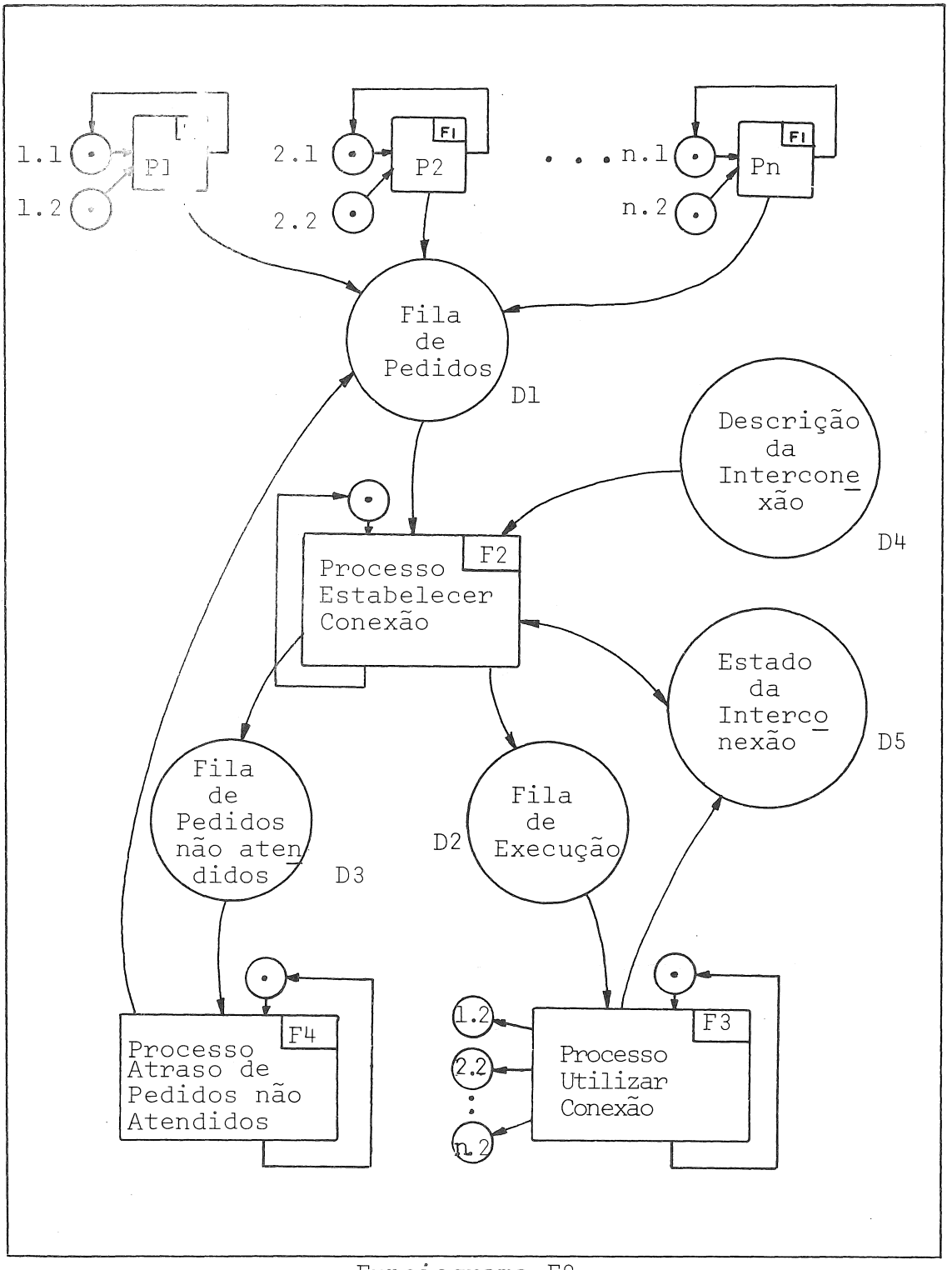
As áreas de dados são elementos estáticos, isto é, armazenam informações que podem ser acessadas pelas estações.

Os canais de aviso estabelecem o fluxo de controle entre as estações: uma estação só pode ser executada se os canais de aviso na sua entrada estiverem marcados. Uma estação sinalizada, após sua execução, coloca uma marca nos canais de aviso da sua saída.

III. ESTRUTURA E FUNCIONAMENTO DO NÚCLEO DO SIMULADOR

III.1 ORGANIZAÇÃO BÁSICA

A organização básica do simulador pode ser vista no funciograma F0.



Funciograma F0

As estações P1,P2,...Pn representam os processadores; sua estrutura interna será descrita no funciograma F1(seção III.2). A partir de uma situação inicial, isto é, estação habilitada, o processador gera um PEDIDO (o processador P1 deseja se comunicar com a memória Mj).

Em paralelo com os processos que representam os processadores, corre o processo ESTABELECER CONEXÃO, descrito em detalhes no funciograma F2(seção II.3). Este processo, basicamente, é um laço onde são retirados pedidos da área D1 (fila de pedidos) e, para cada PEDIDO, é feita a tentativa de estabelecer um caminho entre o processador Pi (identificado no PEDIDO) com a memória Mj. Se for possível estabelecer o caminho Pi-Mj, o conjunto de arestas que o compõe é colocado na área D2 (fila de execução). Se, por outro lado, não é possível, no momento, estabelecer Pi-Mj o próprio PEDIDO é colocado na área D3 (fila de pedidos não atendidos). As áreas D4 (descrição do esquema de interconexão) e D5 (descrição do estado) são consultadas no momento de estabelecer o caminho.

Em paralelo com os processos acima, corre o processo UTILIZAR CONEXÃO, descrito em detalhes no funciograma F3(seção III.4). Este processo é um laço onde se simula a utilização do caminho estabelecido entre um processador e uma memória. Após esta utilização é liberado o caminho (acesso a área D5), permitindo seu uso por outros pedidos. A utilização do caminho (troca de dados entre o processador P1 e a memória Mj) é simulada retardando sua liberação por um tempo IM1, que acompanha o pedido desde a sua geração. Após, o processador Pi é sinalizado, podendo enviar novo PEDIDO.

O processo PEDIDOS NÃO ATENDIDOS também corre em paralelo com os já descritos e é detalhado no funciograma F4(seção III.5). Sua função é a de recolocar os pedidos não atendidos por falta momentânea de recursos na área D1 para que o processo ESTABELECER CONEXÃO possa, numa nova volta do laço, tentar atendê-lo.

Estando estes processos em paralelo, e acessando áreas comuns, é necessário que exista exclusão mútua sobre certas áreas:

Área D1: fila de pedidos - acessada para escrita pelos processos P1,P2,...Pn e pelo processo PEDIDOS NÃO ATENDIDOS; acessada para leitura pelo processo ESTABELECER CONEXÃO. Esta área é estruturada como uma fila circular em alocação seqüencial, e para a sua manipulação existem um algoritmo de inserção e um de retirada; qualquer acesso será feito apenas através destes algoritmos. O número M de nodos será igual ao número de processadores Pi, i=1,2,...N; neste caso a estrutura nunca entrará em overflow; se esta situação tiver que ocorrer no estudo de um caso, a fila terá que ser redimensionada, tronando M menor que N. Pode ocorrer a situação de underflow. Um nodo da fila, chamado PEDIDO, tem a estrutura lógica apresentada na figura 4.

Todos os campos acima são montados em tempo de geração do

pedido pelo próprio processador Pi.

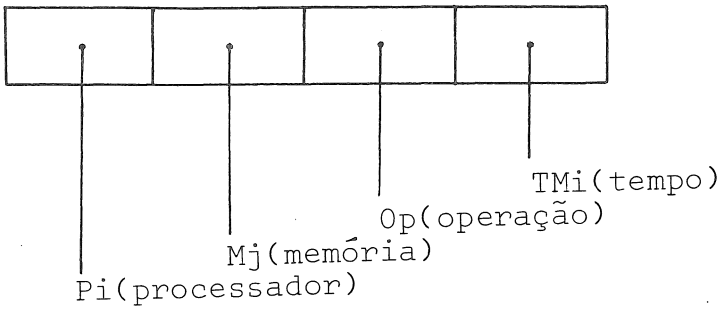


Figura 4 - Estrutura lógica de um PEDIDO

Área D2: fila de execução - acessada para escrita pelo processo ESTABELECER CONEXÃO; acessada para leitura pelo processo UTILIZAR CONEXÃO. Esta área foi estruturada como fila e para sua manipulação existem algoritmos de inserção e de retirada. A estrutura é a mesma da área D1. A estrutura do nodo, entretanto, será diferente: além dos campos descritos na figura 4 existe um novo campo, de ELO, necessário para a liberação do caminho após sua utilização; esta informação é criada internamente ao processo ESTABELECER CONEXÃO.

Área D3: fila de pedidos não atendidos - acesso para escrita pelo processo ESTABELECER CONEXÃO; acesso para leitura pelo processo PEDIDOS NÃO ATENDIDOS. Manipulada através de algoritmos de inserção e retirada em filas. É idêntica à área D1.

Área D5: estado da interconexão - esta área reflete a utilização dos barramentos durante uma simulação. É uma área protegida contra acesso simultâneo pelos dois processos que podem modificá-la: ESTABELECER CONEXÃO e UTILIZAR CONEXÃO. A área tem um papel dinâmico, refletindo a alocação e liberação de segmentos de barramento para o atendimento de pedidos.

Dentro do processo ESTABELECER CONEXÃO o acesso a D5 é feito pela estação MONTACAMINHO (ver funciograma F2). Esta estação procura segmentos livres de barramentos que estabelecem a ligação entre o processador Pl e memória Mj, especificados no pedido retirado da fila. Quando o caminho for estabelecido, os segmentos de barramentos utilizados estarão marcados como "não disponíveis" dentro da área D5.

Na estação UTILIZAR CONEXÃO o acesso a esta área é feito pela estação LIBERACAMINHO (funciograma F3) que libera todos os segmentos de barramentos utilizados para estabelecer o caminho Pi-Mj marcando-os como "disponíveis".

Uma única área não necessita exclusão mútua:

Área D4: descrição da interconexão - esta área é estática,

criada na inicialização do sistema e não mais modificada. Ela descreve a representação interna da interconexão, mostrando as ligações processadores-arbitradores-memórias. O primeiro passo na simulação do comportamento de uma interconexão será o de gerar a descrição interna desta interconexão. No futuro isto será feito por meio de uma LDH.

A interconexão é descrita como um grafo, onde os nodos representam os elementos processadores, arbitradores e memórias. As arestas representam seções de barramento. Os nodos adjacentes permitidos são: ligações processador-arbitrador, arbitrador-arbitrador e arbitrador-memória.

A representação deste grafo é feita por matriz de adjacência. Nas linhas são definidos os processadores, arbitradores e memórias, enquanto que nas colunas são definidos os arbitradores. Por exemplo, a estrutura mostrada na figura 5 é descrita pela matriz da figura 6.

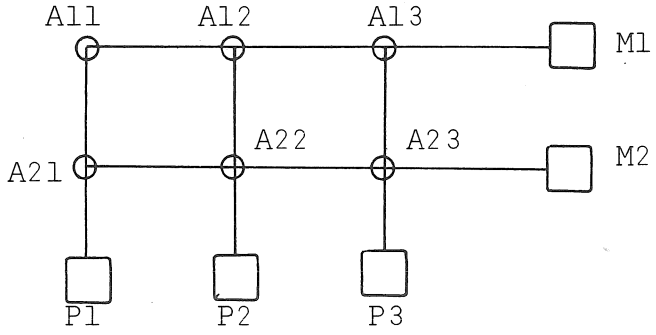


Figura 5 - Esquema de Interconexão

	P1	P2	P3	A11	A12	A13	A21	A22	A23	M1	M2
A11											
A12											
A13											
A21											
A22											
A23											

Figura 6 - Matriz de adjacência para a interconexão

A visão seqüencial do sistema simulador descrito no funciograma F0 pode ser obtida:

1. Um processador Pi gera um pedido onde está especificada uma memória e uma operação (escrita/leitura).

2. O pedido é inserido na área D1.

3. O processo ESTABELECER CONEXÃO retira o pedido da fila e tenta montar a conexão entre o processador e a memória especificados no pedido, utilizando informações das áreas D4 e D5.

4. Se o caminho é estabelecido, este é colocado na área D2.

5. Se o caminho não pode ser estabelecido, o pedido original será inserido na área D3.

6. O processo UTILIZA CONEXÃO retira um pedido com conexão já estabelecida, da área D2 e simula a operação esperando um tempo Ti antes de liberar o caminho, utilizando o parâmetro e, que aponta para a lista dos segmentos utilizados na conexão (atualização da área D5) e avisar o processador Pi.

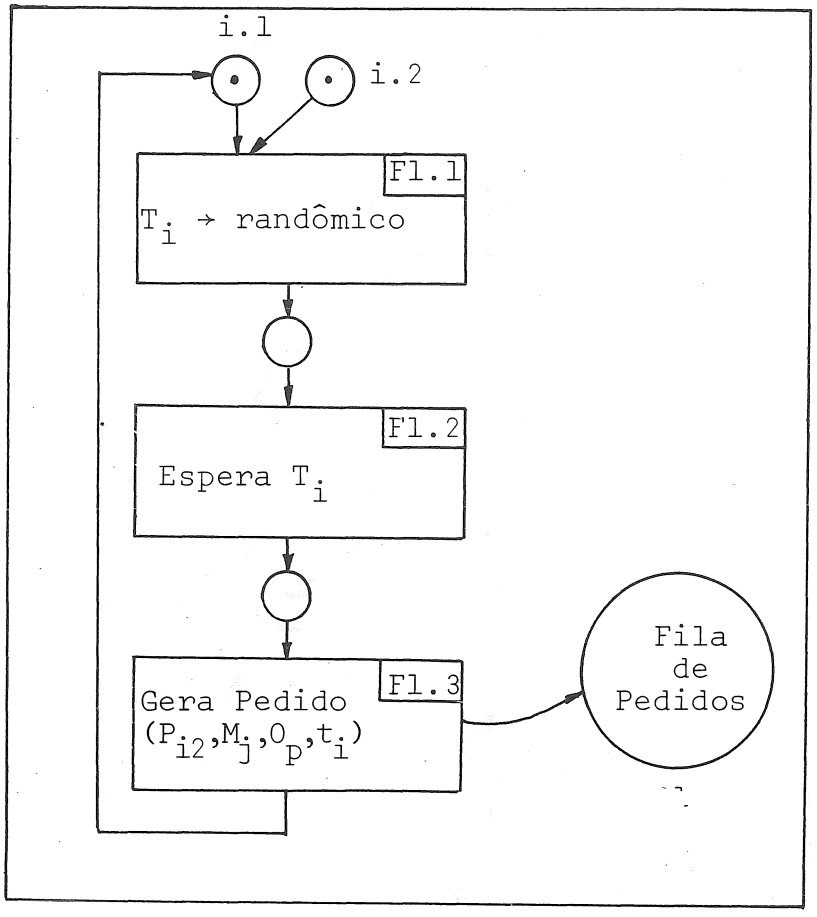
7. O processo PEDIDOS NÃO ATENDIDOS retira um pedido da área D3 e o insere novamente na área D1.

III.2 ESTRUTURA DE UM PROCESSADOR PI

Cada processador Pi do funciograma geral (F0) é representado por uma estrutura igual à descrita no funciograma F1.

Inicialmente ambos os canais de aviso I.1 e I.2 estão sinalizados, permitindo o disparo do processo. Após o disparo e execução das três estações internas, a estação "gera pedido"(F1.3) coloca a marca novamente no canal I.1, mas o processo não está habilitado até que a marca I.2 seja também colocada; isto será feito pela estação F3 do funciograma F0, após o atendimento do pedido.

A estação F1.1 coloca numa variável Ti um valor randômico; este valor será utilizado para simular tempo de processamento, isto é, o processador Pi, durante este tempo não envia nenhum pedido. Após esta espera o processador gera novo pedido de leitura/escrita na estação F1.3. O pedido tem a estrutura descrita na figura 4 gerando a quádrupla (Pi,Mj,Op,TMi). O TMi tem a função de simular a quantidade de dados sobre a qual a operação Op, leitura/escrita, é realizada. Na estação UTILIZAR CONEXÃO do funciograma F0 os segmentos de barramento utilizados ficarão marcados como "não disponíveis" durante este tempo TMi. Finalmente, inclui-se o pedido gerado na fila D1 do funciograma F0 de onde será retirado por ESTABELECER CONEXÃO.



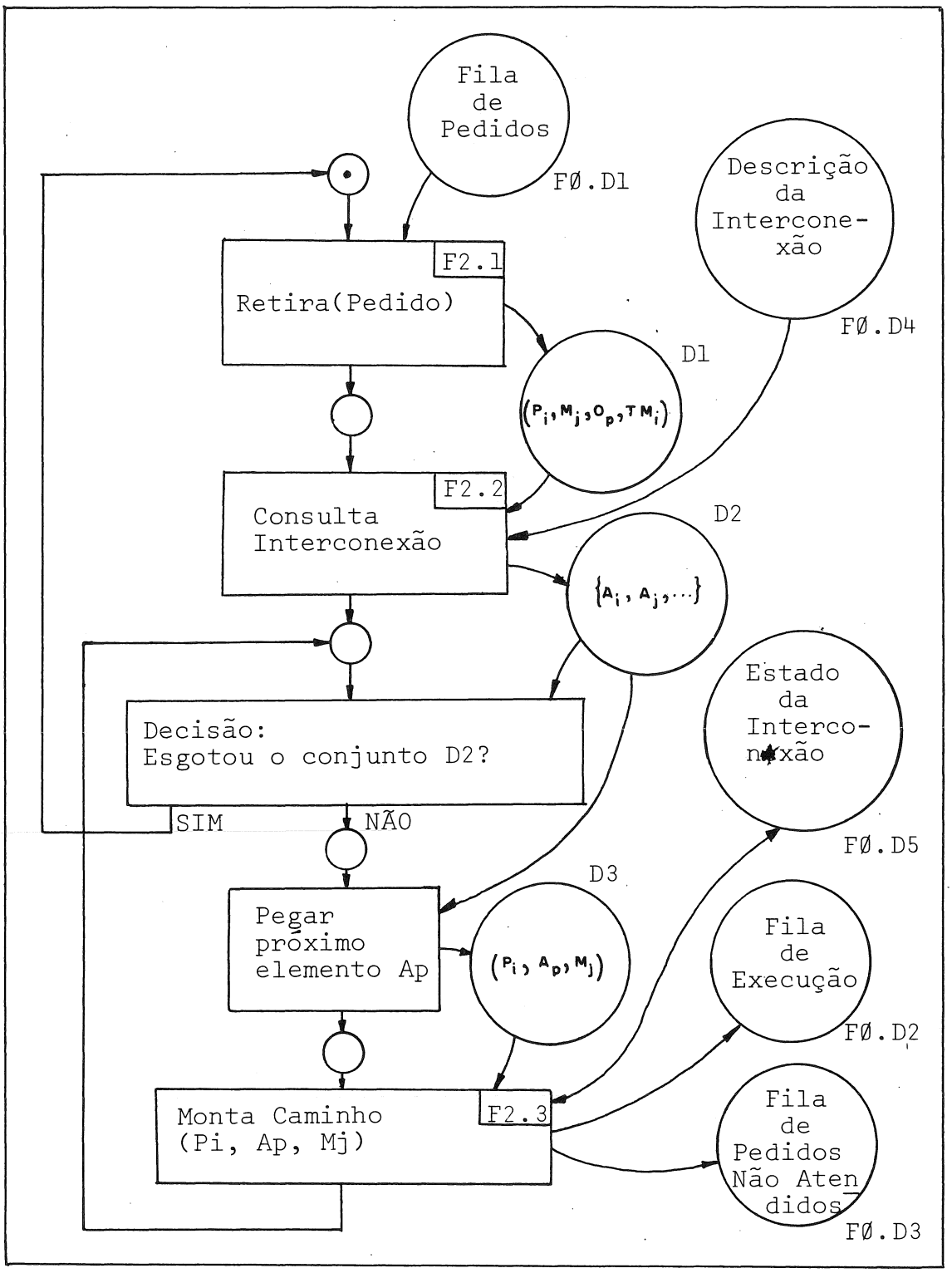
Funciograma Fl

III.3 ESTABELECIMENTO DA CONEXÃO

O processo descrito no funciograma F2 é o responsável pela tentativa de estabelecimento do caminho Pi-Mj. Se esta for bem sucedida, o caminho montado será inserido na fila de execução. Caso contrário, o pedido original é colocado na fila de pedidos não atendidos.

Inicialmente a estação F2.1 está habilitada. Ela retira um pedido da área F0.D1 onde estão os pedidos gerados pelos processadores. A estação F2.2 - Consulta interconexão - verifica quê arbitradores, ligados ao processador Pi, podem ser utilizados para estabelecer uma conexão entre Pi e Mj; o conjunto destes arbitradores é colocado na área D2, definindo um conjunto controlador de laço.

A estação F2.3 - Montacaminho - tenta estabelecer uma ligação entre Pi e Mj, utilizando arbitradores que tomarão suas decisões baseados nas informações de estado de segmentos de barramentos, descritos na área F0.D4. Como resultado é colocada na área F0.D2 a seqüência de segmentos de barramento utilizados para a montagem do caminho.



Funciograma F2

Na tentativa de montar Pi-Mj leva-se em conta o estado atual dos segmentos de barramento na interconexão (figura 7). Os parâmetros de entrada para montacaminho são: Pi,Ap,Mj. O parâ-

metro de saída é o ponteiro E (elo) para o cabeça de lista do caminho montado; este endereço será necessário para executar a liberação do barramento (ver F3).

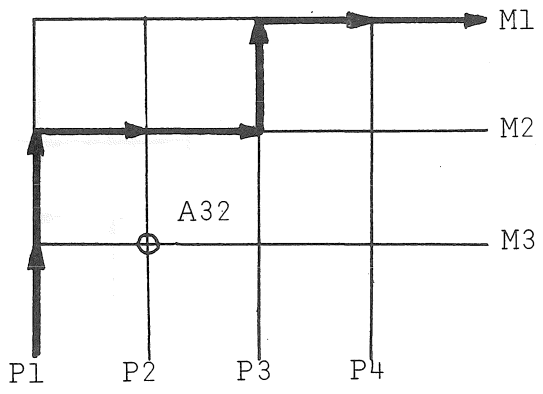
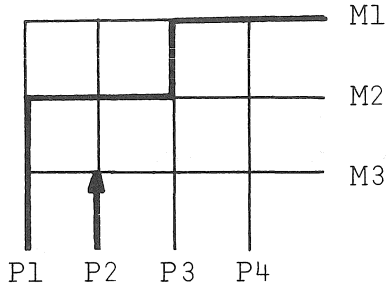


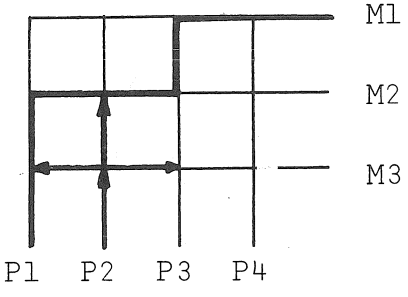
Figura 7 - Interconexão com P1-M1 estabelecido

Se for necessário estabelecer P2-M2, com a interconexão no estado descrito pela figura 7, MONTACAMINHO é ativada com os parâmetros P2, A32, M2 (figura 8a).

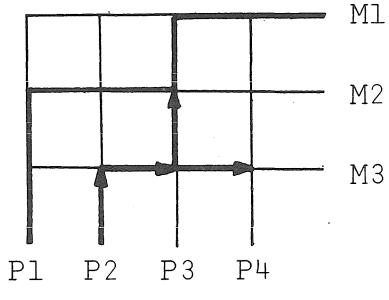
MONTACAMINHO verifica que segmento de barramento está livre, a partir de A32, o marca como "NÃO DISPONÍVEL" e continua o processo a partir do novo arbitrador Aij. O processo de busca do segmento livre pode ser realizado de diversas formas. Por



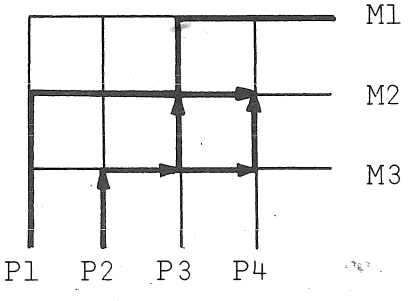
a)



b)



c)



d)

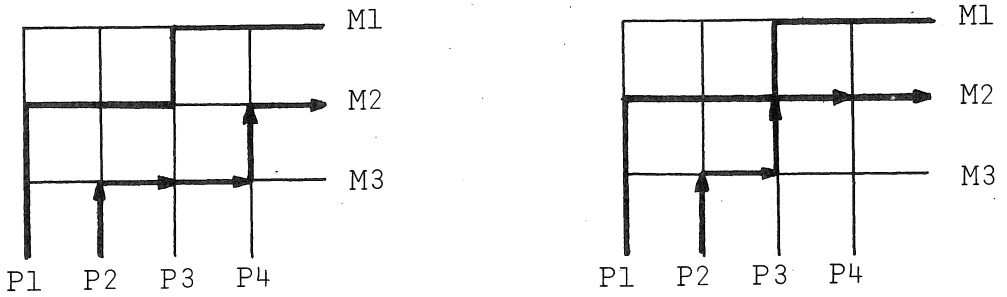


Figura 8 - Montagem do caminho P2-M2

exemplo, diversos caminhos podem ser testados em paralelo e a primeira tentativa bem sucedida desativa todos os outros processos, liberando os segmentos utilizados nas tentativas não sucedidas. Dentro desta filosofia, o processo iniciado na figura 8a teria a seqüência mostrada na figura 8b a 8f.

Observa-se que a partir da situação descrita na figura 8d pode se chegar tanto à situação (e) como à situação (f), na dependência do processo que primeiro tiver acesso ao segmento do barramento A24-M2, marcando-o como "não disponível" e eliminando portanto o recurso para o outro processo.

III.4 USO DA CONEXÃO

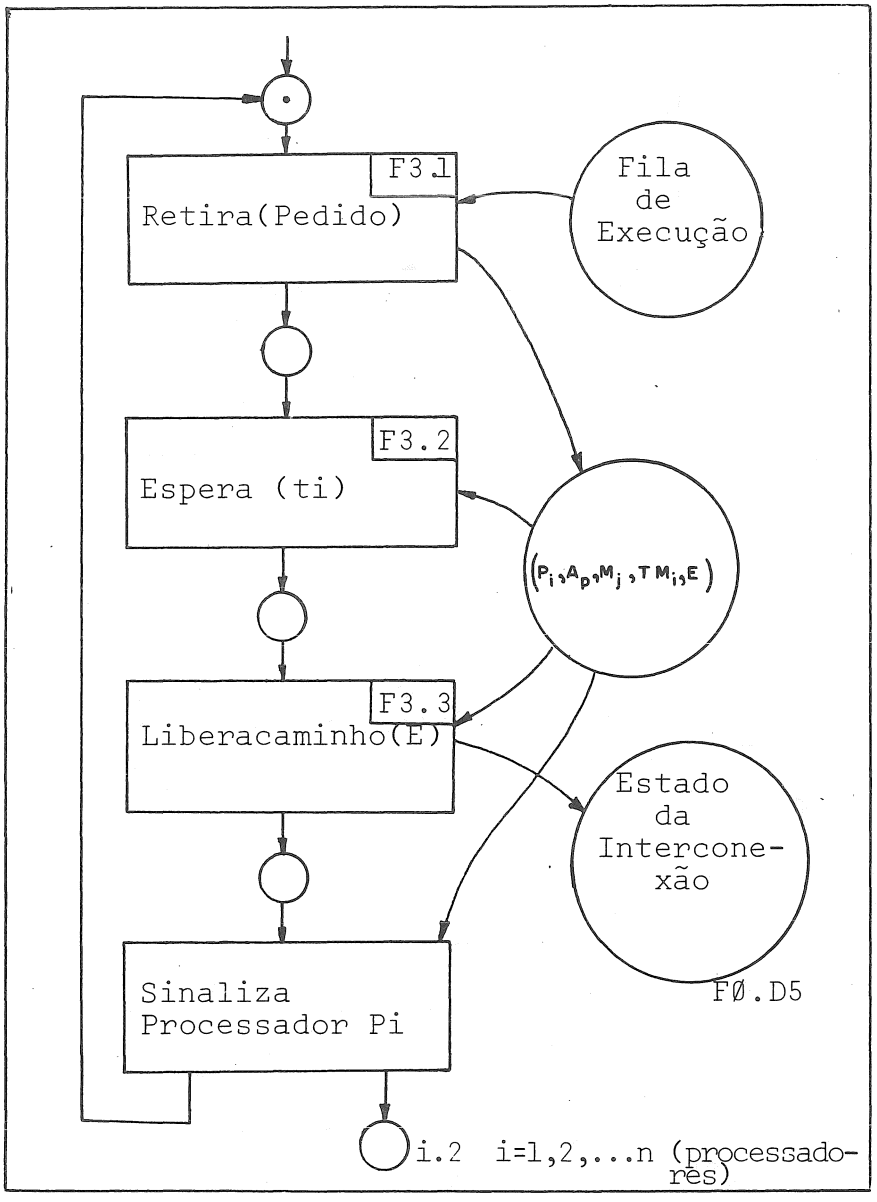
O processo descrito no funciograma F3 simula a utilização do caminho Pi-Mj.

Inicialmente é retirado um pedido da fila de execução; pode ocorrer underflow, fazendo com que o processo seja suspenso. No pedido retirado, o parâmetro TMi simula a utilização do caminho. Os segmentos de barramento que o compõem permanecem marcados como "não disponíveis" durante este tempo TMi.

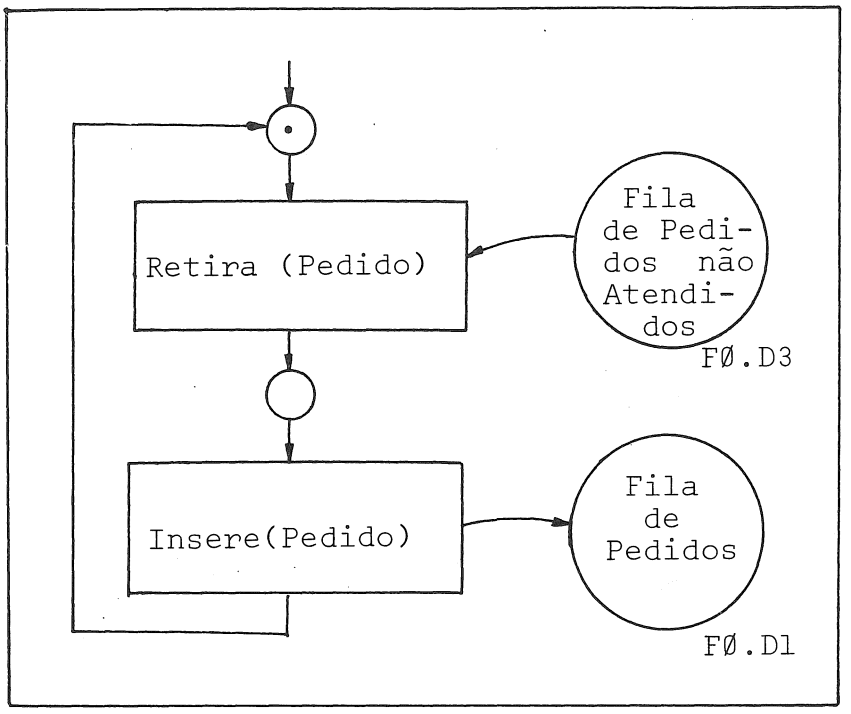
A estação F3.3 - Liberacaminho - utiliza o parâmetro E do pedido (Pi,Mj,Op,TMi,E) para marcar os segmentos de barramento como "disponíveis". Desde a geração do pedido pelo processador Pi até a liberação do caminho, o processador permanece inativo.

III.5 PEDIDOS NÃO ATENDIDOS

O processo F4 tem a função de retirar pedidos da fila F0.D3, que contém os pedidos que não foram atendidos pela estação montacaminho, e inseri-los novamente na fila F0.D1-fila de pedidos, para uma nova tentativa de estabelecer a conexão.



Funciograma F3



Funciograma F4

IV. CONCLUSÕES

O SSIP é uma ferramenta de grande flexibilidade para o arquiteto de sistemas de múltiplos processadores, permitindo o estudo e a avaliação de diferentes interconexões. Como tal, ele pode ser utilizado não só na análise de interconexões para multiprocessadores como também na de redes de computadores ou de sistemas distribuídos.

Atualmente o SSIP está sendo desenvolvido para a definição da arquitetura mais viável do projeto SMM. Após esta etapa a ferramenta será generalizada permitindo o seu uso em ensino e pesquisa. A experiência ganha nesta primeira etapa,fornecerá os subsídios para melhorar o SSIP e implementar uma LDH que, utilizando o núcleo básico definido, permitirá definir diferentes estruturas de interconexão e arbitradores para uma dada aplicação.

Não há dúvida que os sistemas de múltiplos processadores terão no futuro uma participação importante - e crescente - no mercado de máquinas. É portanto imprescindível o desenvolvimento de ferramentas tais como o SSIP que permitam o estudo aprofundado da organização de tais máquinas e testes conclusivos de desempenho relativo entre diversos esquemas de interconexão.

O SSIP está atualmente na etapa final de sua definição devendo entrar em fase de programação.

BIBLIOGRAFIA E REFERÊNCIAS

(And.75) ANDERSON, G.A. E JENSEN, E.D.
"Computer inter-connection structures: Taxonomy,Characteristics and examples", Computer Surveys, v7, n4, dec75, p197-213.

(Bar.75) BARBACCI, M.R.
"A Comparison of register transfer languages for describing Computer and digital systems", IEEE Transaction on Computer, feb75. p137-150.

(Ens.75) ENSLOW, P.H.
"Multiprocessor architecture - a survey", 1975 Sagamore Computer Conference on Parallel Processing, p63-70.

(Har.77) HARTENSTEIN, R.W.
"Fundamentals of structured hardware design". North Holland 1977.

(Kle.79) KLEIN, D.D.
"MMPS - a reconfigurable multi-microprocessor simulator system". Afips-ncc79, p199-203.

(Nil.71) NILSSON, N.J.
"Problems-solving methods in artificial intelligence". New York, McGraw-Hill, 1971.

(Ric.77) RICHTER, G.
"O sentido e o valor do banco de dados", Dados e idéias, n6, jun/jul77, p2-14.

(Thu.78) THURBER, K.J.
"Computer Communication Techniques", Computer architecture news, ACM-Sigarch, v7, n3, oct78, p7-16.

(Tri.79) TRIPATHI, A.R. & LIPOVSKI, G.J.
"Packet switching in banyan networks", 6th annual computer architecture, april 79, p160-167.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

METODOLOGIA PARA DETERMINAR EL RENDIMIENTO DE UN COMPUTADOR Y DE SU AMPLIACION

Daniel H. Mirol
INSTELCOM S.A.
Buenos Aires, Argentina

RESUMEN

Los plazos prolongados en la entrega de computadoras, y la gran cantidad de empresas que se han volcado a la informática, pueden producir que un equipo quede saturado en poco tiempo. En ese momento se solicita su ampliación, sin poder realizar un estudio con la profundidad requerida y con los problemas de brindar mal servicio: entrega de información atrasada, eventual pérdida de clientes, poca o ninguna posibilidad de recuperación ante inconvenientes, etc.

Se propone una metodología de análisis que, en este caso, se desarrolló para conocer el rendimiento probable de un computador que aún no había sido recibido, y para elaborar una estrategia de crecimiento de computador basada en costos y tiempos de producción esperada, utilizado para ello el modelo de Ramificación y Acotación (BRACH AND BOUND), en la forma de un Arbol de Decisiones.

INTRODUCCION

Este trabajo fue realizado ante la necesidad de contar con información sobre el probable desenvolvimiento del computador NCR 8230, tomando como base un sistema tipo y suponiendo que n cantidad de usuarios lo usarían.

La tarea se encaró en condiciones de total incertidumbre sobre el rendimiento del computador, que aún no había sido recibido, y sobre el sistema que fue comprado en forma de "paquete" y no se contaba con suficiente información.

Los datos que se pretendía obtener son los siguientes:

a) Evaluación de tiempos y costos de la configuración actual.

b) Cantidad de usuarios que se puede procesar con la configuración actual.

c) Filosofía de crecimiento del equipo y sus costos.

El estudio se encaró en tres etapas:

1- Determinación de los parámetros característicos de un programa

2- Hipótesis sobre el sistema tipo.

3- Perfomance esperada del equipo, ampliación y costos.

PARAMETROS CARACTERISTICOS DE UN PROGRAMA

2.1. Memoria

Este dato se debe obtener de los listados de compilación.

2.2. Espacio de disco

Los programas requieren dos tipos de espacios en disco:

a) el que ocupa el programa en si mismo.

b) el que ocupa sus archivos y que logicamente depende del número de ellos y su volumen.

2.3. Tiempo de procesador

Se calculará con un valor del 10% del tiempo total promedio de Entrada/Salida (E/S)

2.4. Tiempo de E/S

Depende del número de transacciones a procesar y del tiempo estimado de acceso y transferencia.

a) Tiempo estimado de acceso:

- Posición de la cabeza promedio	0,0350 SEG.
- Latencia promedio	0,0125 SEG.
- Tiempo promedio de acceso	0,0475 SEG.

b) Velocidad de transferencia

Como la transferencia se efectúa por sectores, la velocidad es:

Velocidad de transferencia de un sector

$$\frac{512}{312.500} = 0,0016 \text{ SEG.}$$

c) Tiempo promedio de acceso y tranferencia (TPAT)

TPAT = 0,0475 + 0,0016 = 0,0491 SEG.

Además de estos tiempos existen otros que ocupa el sistema operativo en:

- Ejecución del RUNTIME INTERPRETER.
- Interpretación y ejecución de comandos.
- OPEN y CLOSE de archivos.
- Manejo lógico de sectores.
- Etc.

Estos tiempos se tienen en cuenta implicitamente si suponemos que el TPAT se utiliza por cada registro lógico a procesar.

Por lo tanto el tiempo promedio de E/S (TPES) será:

$$TPES = \frac{N° \text{ de REGISTROS} \times 0{,}0491}{60}$$

2.5. Tiempo de impresión

Dependerá del número de registros a ser impresos; se aplicará la siguiente fórmula:

$$TI = \frac{N° \text{ REGISTROS}}{250}$$

Se toma 250 l/m como velocidad promedio ya que el fabricante declara 300 l/m, de esta manera se tiene en cuenta:

- El retardo por búsqueda de caracteres especiales.
- Tiempos de cambio de papel
- Etc.

3. HIPOTESIS SOBRE EL SISTEMA TIPO

3.1. En base al análisis de la descripción del sistema y a entrevistas con sus constructores se lograron los siguientes valores:

a) Tamaño promedio de programas:	13 K
b) Cantidad total de Registros:	20.000
c) Tamaño promedio de Registros:	200
d) Cantidad total de archivos:	10
e) Cantidad total de líneas de impresión:	50.000
f) Cantidad promedio de archivos por programas:	3
g) Uno de estos es una salida impresa	
h) Cantidad total de programas:	60

i) El 40% de los programas son diarios y el resto mensuales — 24 y 36

j) El 20% del total de líneas de impresión es diario, el resto mensual — 10.000 y 40.000

3.2. ESTIMACIONES DE TIEMPOS Y ESPACIOS

3.2.1. Tiempo promedio de ejecución de un Programa

a) TIEMPO DE E/S

Para calcular este tiempo se deberá tomar la cantidad total de registros que procesará el programa.

$$TPES = \frac{\frac{20.000}{10}}{60} \times 3 \times 0{,}0491 = 5 \text{ minutos}$$

Este valor es el más probable, pero para los siguientes cálculos se tomará el más crítico:

TPES + 1 = Valor más crítico (TCES)

b) TIEMPO DE PROCESADOR

Se calcula el 10% de la E/S

TIEMPO PROCESADOR (TP) = 6 x 0,10 = 0,6 min.

TIEMPO EJECUCION (TE) = TCES + TP = 6 + 0,6 = 6,6

TE $\simeq$ 7 min.

Si el programa es ejecutado en multiprogramación se deberá incrementar el tiempo de ejecución en un 30%

TE MULTIPROGRAMACION = (TEM) $\simeq$ 9 min.

3.2.2. Tiempo Promedio de Impresión de la salida de un programa

Según las hipótesis, uno de los tres archivos es de impresión; por lo tanto:

$$TI \simeq 8 \text{ min.}$$

3.2.3. Tiempo promedio de ejecución del sistema

a) CORRIDAS DIARIAS

según la hipótesis son 24 programas

$$TED = \frac{24 \times 9}{60} = 3{,}6 \text{ h}$$

b) CORRIDAS MENSUALES

Los programas que restan son 36, por lo tanto:

$$TEM = \frac{36 \times 9}{60} = 5{,}4 \text{ h}$$

3.2.4. Tiempo promedio de impresión de las salidas del sistema

a) CORRIDAS DIARIAS

Según la hipótesis son 10.000 líneas

$$TI = \frac{10.000}{250} = 40 \text{ m.}$$

b) CORRIDAS MENSUALES

$$TI = \frac{\frac{40.000}{250}}{60} \simeq 3 \text{ h.}$$

3.2.5. Espacio en disco

El espacio que ocuparía el sistema tipo estaría dado por el ocupado por:

a) Los programas:

60x 13.000 = 780.000 BYTES

Como es poco significativo y serán compartidos por los usuarios, los programas serán alojados en un disco fijo.

b) Los archivos:

20.000 x 200 = 4.000.000 BYTES

Estos se pueden manejar en un disco removible ya que su capacidad máxima es de 5.000.000 de bytes.

4. ESTUDIO DE LA PERFOMANCE ESPERADA DEL COMPUTADOR NCR 8230

4.1. UTILIZACION DIARIA DEL EQUIPO

Para los procesos diarios se parte de tres variables; turnos de 8, 12 ó 16 horas. A los efectos de determinar los tiempos netos de procesos para usuarios deben restarse:

- Dos horas que se dedican a programación
- Una hora que corresponde al tiempo perdido en la carga del sistema operativo y puesta en régimen del equipo.

Del total de particiones, (4 ó 6, según cantidad de CRT), se debe restar una que se utilizará para:

- Programa listador del Spool.
- Copia de archivos.
- Ejecución de comandos

La relación entre turnos y particiones permite componer seis alternativas:

a) Turno de 8 horas con cuatro particiones

tp1 = (8-2-1) x (4-1) = 15; | tp1 = 15 |

b) Turno de 8 horas con seis particiones

tp2 = (8-2-1) x (6-1) = 25; | tp2 = 25 |

c) Turno de 12 horas con cuatro particiones.

El tiempo en horas sumando todas las particiones disponibles para producción sería:

tp3 = (12-2-1) x (4-1) = 27; | tp3 = 27 |

d) Turno de 12 horas con seis particiones

tp4 = (12-2-1) x (6-1) = 45; | tp4 = 45 |

e) Turno de 16 horas con cuatro particiones

tp5 = (16-2-1) x (4-1) = 39; | tp5 = 39 |

f) Turno de 16 horas con seis particiones

tp6 = (16-2-1) x (6-1) = 65; | tp6 = 65 |

4.2. ANALISIS DE LA PRODUCCION

Para realizar el análisis de la producción esperada del equipo se toma como unidad de medida el Sistema Tipo. Según las estimaciones analizadas en el punto 2 se deben contemplar las corridas de fin de mes.

Para este análisis se tendrá en cuenta la ampliación del sistema que producirá las siguientes variaciones:

- Corridas diarias de 24 a 30 programas
- Corridas mensuales de 36 a 50 programas

En el cuadro 1, se puede ver un resumen en forma acumulada hasta ocho usuarios de los tiempos de proceso e impresión. Se tienen en cuenta la duración de un programa, la cantidad de ellos y los tiempos de impresión.

En el gráfico 1 se combinan los datos del cuadro 1 y los valores obtenidos en 4.1. En éste se puede determinar que cantidad de usuarios se pueden procesar según la alternativa de turno y particiones y en monoprogramación equivalente.

CUADRO 1

TIEMPOS DE PROCESO E IMPRESION

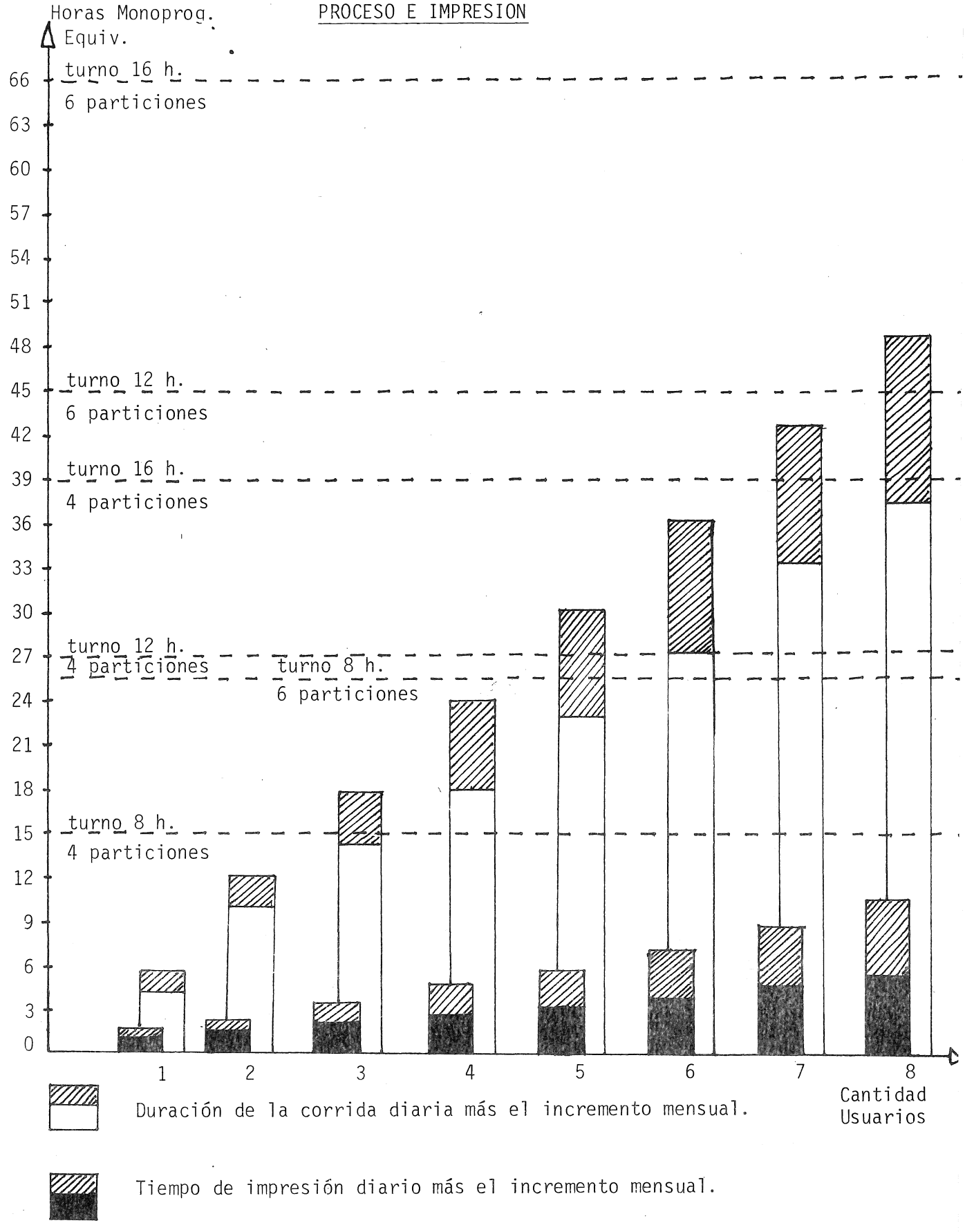
Cantidad de usuarios	Cantidad de Programas Totales acumulados			Tiempos de Proceso (Totales acumulados*)		Tiempos de impres. Totales acumulados		
	Corrida diaria	Corrida mens. (Total /5)	Diar+mens.	Corr.Diar.	Corrida diar+mens.	Diar	Mens.	Diar+Mens.
1	30	10	40	4,5	6	0,7	0,6	1,3
2	60	20	80	9	12	1,4	1,2	2,6
3	90	30	120	13,5	18	2,1	1,8	3,9
4	120	40	160	18	24	2,8	2,4	5,2
5	150	50	200	22,5	30	3,5	3,0	6,5
6	180	60	240	27	36	4,2	3,6	7,8
7	210	80	280	31,5	42	4,9	4,2	9,1
8	240	90	320	36	48	5,6	4,8	10,4

* Total Horas = Duración Programa x Cantidad Programas

Duración Programa = 0,15 Horas.

GRAFICO 1

GRAFICO DE TIEMPOS DE PROCESO E IMPRESION



4.3. CRITERIOS DE PRODUCCION

a) Turno

Se tomaría como turno el de 16 horas; este permitiría tener un margen para recuperación en caso de falla del computador,aire acondicionado, etc.

b) Horas de procesamiento diario

Se tomaría como valor máximo de horas de procesamiento diario el 80% del total de horas disponibles en monoprogramación equivalente. Este permitiría absorber las variaciones en los tiempos previstos para la carga diaria.

c) Arrendamiento de la segunda impresora.

La incorporación de otra impresora sucedería cuando se esté procesando la cantidad de horas correspondiente a cinco sistemas tipo. Esto permitiría continuar con el servicio en caso de falla de alguna de las impresoras, aire acondicionado, etc.

d) Máximo de Sistemas en ejecución simultánea

El número de tareas en multiprogramación será igual a la cantidad de ejes disponibles, ya que cada sistema ocupará un disco, memos uno que estaría ocupado con:

- Sistema operativo
- Archivos de trabajo
- Programas
- Etc.

4.4. COSTOS ACTUALES Y DE AMPLIACION

Tomando como base los criterios expuestos en los puntos ante-

riores se proyectaron los costos de las distintas alternativas los cuales se exponen a continuación:

a) COSTOS VARIABLES

CUADRO 2

	TURNO 8 Hs.			TURNO 12 Hs.			TURNO 16 Hs		
Alquiler equipos CONTRATADOS	Arr.	Mant.	Total	Arr.	Mant.	Total.	Arr.	Mant.	Total
1 Proces. con 96k	533	170		666	255		799	340	
2 CRT	185	77		231	115		277	154	
2 Unid. de disco	635	194		794	291		952	388	
1 Impresora	544	193		680	289		816	386	
1 7500	154	62		192	93		231	124	
Operador 8200			787			787			1.574
TOTAL			3534			4393			6.041
Alquiler equipos A CONTRATAR									
Memoria 32 k	120	54		150	81		180	108	
Adaptación a 8250	110	-		137	-		165	-	
1 CRT	92	38		115	57		138	76	
TOTAL			414			540			667
Alquiler equipos VARIOS									
1 Impresora	544	193	737	680	289	969	816	386	1.202
1 Unidad de disco	317	97	414	396	145	541	475	194	669
TOTAL GENERAL			5099			6443			8.579

OTA: El arrendamiento básico se incrementa para un uso mayor de ocho horas

b) COSTOS FIJOS

CUADRO 3

Personal a Costo Fijo	Total $ x 1000	Total u$s
Analista SCD	900	
Analista-Programador	750	
Implementador	720	
Operador 7200	330	
Total Remuneraciones	2.700	3.375

Los costos fijos están formados por:

Personal a Costo Fijo:	u$s 3.375.-
Alquiler y gastos varios:	u$s 1.300.-
Total	u$s 4.675.-

4.5. ANALISIS DE LA AMPLIACION DEL EQUIPO

a) Capacidad de producción con configuración actual.

Con el turno de 12 horas previsto y cuatro particiones, se pueden procesar hasta cuatro usuarios según se ve en el gráfico 1. No obstante como cada sistema ocupa un disco, y de los cuatro disponibles para usuarios uno no lo está, quedarían sólo tres ejes.

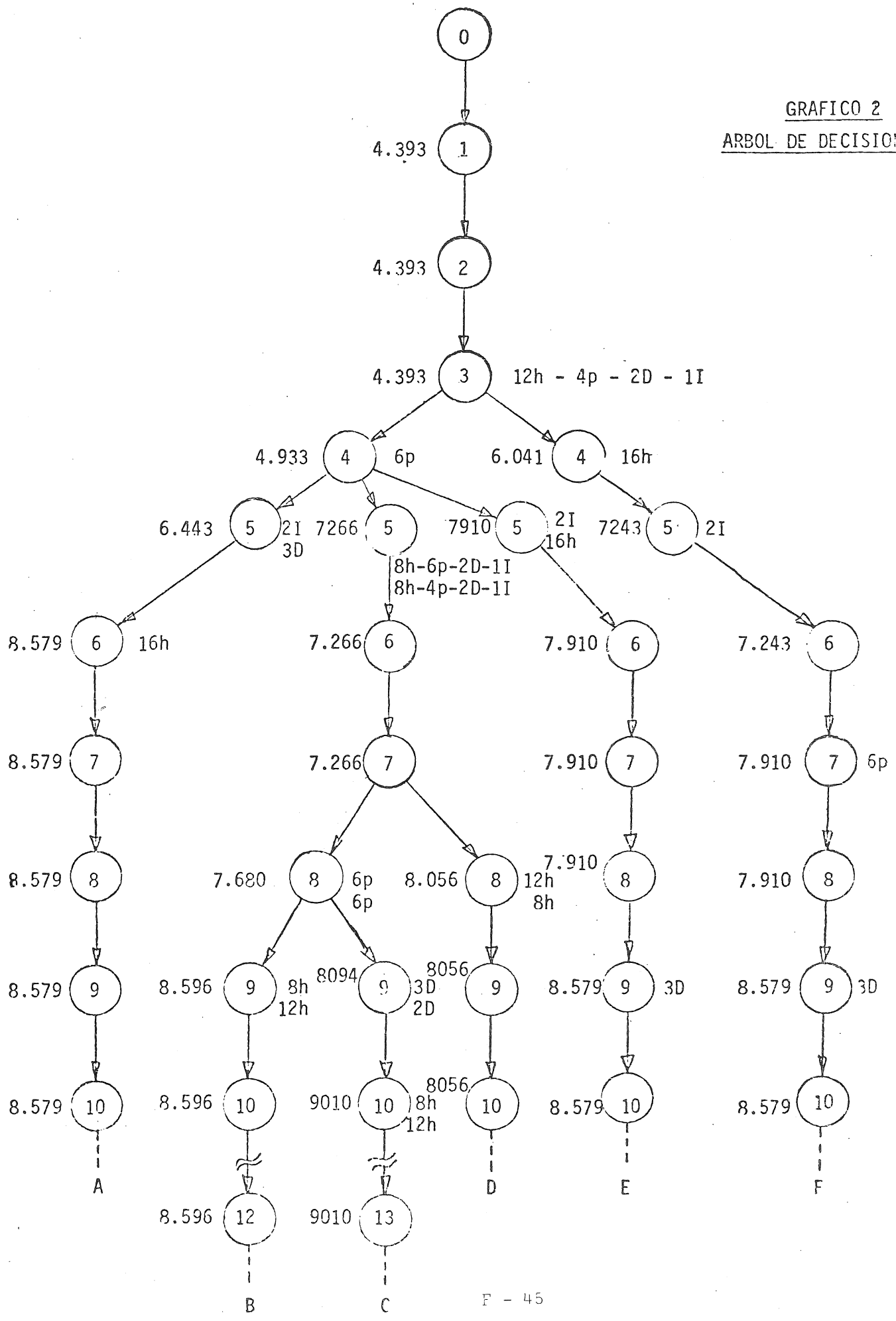
Por lo tanto con la configuración actual se podrían procesar hasta tres usuarios.

b) Ampliación del equipo.

Para realizar este análisis se utilizó como base el modelo de Ramificación y Acotación (Branch And Bound) en la forma de un Arbol de Decisiones. (GRAFICO 2).

GRAFICO 2

ARBOL DE DECISION



Las ramificaciones se producen cuando el equipo no tiene capacidad para procesar un usuario más y cada ramificación representa una alternativa de ampliación de la capacidad productiva.

La acotación surge al elegir la ramificación de menor costo o de mayor conveniencia, según el caso.

c) " ARBOL DE DECISIONES "

La simbología utilizada es la siguiente:

(n) : es el proceso de n cantidad de usuarios

⟶ : es el incremento en un usuario

El número que aparece a la izquierda de (n) es el costo de arrendamiento de los equipos más el sueldo del operador, todo en dólares.

A la derecha de (n) se especifican los cambios, en las horas de los turnos y el agregado de nuevas máquinas, necesarias para procesar los n usuarios. Las abreviaciones utilizadas son:

8 h / 12 h / 16 h - 8, 12 ó 16 horas de turno

4 p / 6 p - 4 ó 6 particiones

2 D / 3 D - 2 ó 3 unidades de dos discos

1 I / 2 I - 1 ó 2 impresoras

Se puede observar en el árbol que hay cuatro puntos de decisión:

a) Para procesar cuatro usuarios se debe decidir:

- Aumentar el turno a 16 horas.
- Aumentar a 6 particiones.

Se elige esta última por ser la más económica.

b) Para procesar cinco usuarios las alternativas de decisión son:

- Agregar una unidad de disco y una impresora
- Contratar otro computador y trabajar en turnos de 8 horas.
- Aumentar el turno a 16 horas y agregar una impresora,

A pesar que la primera es la de menor costo, se elige la segunda ya que con el agregado de un usuario más, ésta pasa a ser la más económica.

c) Para procesar ocho usuarios las opciones son:

- Aumentar el segundo computador a seis particiones.
- Aumentar el turno en el primer computador a 12 horas.

Se elige la primera por ser la de menor costo.

d) Para procesar nueve usuarios las decisiones son:

- Aumentar el turno a 12 horas en el segundo computador.
- Agregar otra unidad de disco al primer computador.

Esta última es la más económica, pero al agregar otro usuario, se debe elegir la primera ya que pasa a ser la de menor costo.

Tomadas las cuatro decisiones analizadas como las más convenientes, queda determinada la rama B como la óptima.

5. CONCLUSIONES

A través de la metodología utilizada, se pudo ampliar el horizonte de información sobre la utilización probable del computador con su configuración actual y su futura ampliación.

Los datos obtenidos prodrán utilizarse para tomar decisiones estrategicas, sobre la política de comercialización de las horas de service y prever la contratación de ampliaciones con tiempo suficiente para que ésta sea oportuna.

De la secuencia óptima de decisiones que determinó la rama "B" como la más conveniente, se puede extraer la siguiente conclusión: Existe un punto, en este caso cuando se procesan cinco usuarios, en que es conveniente el alquiler de otro computador en vez de continuar con la expansión de uno sólo. Aparte de la conveniencia de costos, existen otros factores que hacen atractiva la operación con dos equipos, algunos de ellos son:

- Menor costo horario
- Mayor margen de ampliación
- Mayor seguridad de servicio
- Mejor distribución de la carga de máquina
- Etc.

A medida que se obtengan datos reales sobre los diferentes parámetros se puede ir replanteando el modelo y seguramente se obtendrán distintos "Arboles de Decisión", que permitirán tomar determinaciones con mayor certeza.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

CAPITULO G

COMPUTACION GRAFICA

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

O USO DE DIALOGOS NA INTERAÇÃO HOMEM–MAQUINA

Liane Margarida Rockenbach Tarouco
Claudia Sabani
Centro de Processamento de Dados
Universidade Federal Do Rio Grande Do Sul
Av. Osvaldo Aranha 99
Porto Alegre, Brasil

OBJETIVO DO TRABALHO

A intensificação do uso de sistemas de processamento de dados interativamente determinou a necessidade de estudar-se diferentes formas para que o diálogo homem-máquina seja estabelecido de uma maneira natural ao homem uma vez que este, se obrigado a manifestar-se de uma forma que não lhe é comum, cometerá muitos erros o que acarretará baixa performance no sistema e pouca confiabilidade nos dados acumulados.

Neste trabalho destacar-se-á a importância da simplificação do diálogo homem-máquina e serão descritos e comentados a luz das experiências desenvolvidas na Universidade Federal do Rio Grande do Sul.

1. INTRODUÇÃO

Na interação com um sistema de processamento de dados através de terminais ocorrem, freqüentemente, problemas devido às características diametralmente opostas dos "contendores" desta interação. De um lado nós temos o ser humano cuja taxa de recepção ou emissão de informações varia de 5 a 50 bits por segundo apenas. Já o computador é capaz de receber, emitir ou processar bits em velocidades de ordem de milhões de bits por segundo. No que tange a erros, a cada 100 caracteres informados pelo ser humano um contém erro. As taxas de erros inerentes aos sistemas automatizados (computador) são bem menores conforme apresentados na figura 1.

No que tange aos aspectos físicos e psicológicos podemos destacar que, se submetido a uma tarefa monótona e repetitiva ou cansativa, o ser humano tende a cometer mais erros, a dispersar sua atenção da tarefa a que é submetido e a impacientar-se. Por seu lado o computador pode ser considerado incansável, é capaz de advertir o usuário de um erro cometido, tantas vezes quantas forem necessárias com a mesma paciência e grau de atenção. Contudo em situações de excessões onde precisa improvisar, é o ser humano que se sai melhor pois toma decisões ante situações inesperadas ao passo que o computador é incapaz de reagir de forma diferente do que foi programado. A capacidade de memorização do ser humano, por sua vez, é bastante limitada, embora ele possa ter acesso a ela por contexto, isto é, a busca se faz seletivamente. O computador pode armazenar muito mais informação do que a memória humana, mas o acesso a ela se faz apenas por uma comparação do tipo "é igual, sim, não", então continua a procurar o que pode levar a um tempo bastante alto até que a informação desejada seja localizada.

Este quadro comparativo nos indica alguns rumos no sentido do desenvolvimento de sistemas que aproveitem ambas as qualidades e complementem as mútuas necessidades. Deve-se buscar formas de diálogo que permitam ao ser humano falar tão próximo do seu modo natural quanto possível para minimizar a quantidade de erros que ele comete. Os sistemas desenvolvidos no computador devem ser capazes de interagir nesta forma com o ser humano detectando e corrigindo todos os erros que ele cometa mesmo que para isso tenha que acessar tabelas, e arquivos para efetuar consistências e outras validações. Há tempo para fazê-lo, uma vez que a velocidade de processamento do computador é muitas vezes maior do que a velocidade de reação do ser humano. Contudo, se o sistema de processamento de dados não for cuidadosamente planejado para apoiar este diálogo, mesmo com a alta velocidade de acesso e de processamento inerente ao computador, o tempo de resposta se tornará demasiado o que prejudicará então a interação homem-máquina.

Os sistemas interativos através de terminal têm sido alvo de atenção e de larga utilização na Universidade Federal

FIGURA 1 - COMPARAÇÃO HOMEM-MÁQUINA

	HOMEM	COMPUTADOR
Velocidade	baixa (30BPS)	alta (M-BPS)
ERROS	10^{-2}	transmissão 10^{-5} periféricos 10^{-9} CPU 10^{-14}
ASPECTOS FÍSICOS E PSICOLÓGICOS	sensível ao cansaço impaciente dispersivo improvisação em situações de exceção	incansável paciência ilimitada atenção aparentemente constante incapacidade de reagir de forma diferente do que foi programado
MEMÓRIA	baixa capacidade acesso por contexto	alta capacidade acesso por comparação

do Rio Grande do Sul, desde 1973. Dos diversos sistemas experimentados adquiriu-se uma considerável experiência que permitiu o aperfeiçoamento de diversas formas de diálogos de modo a se atingir nos sistemas atuais mesmo usuários leigos e escassamente treinados. Neste trabalho serão discutidos e comentados as diversas formas de diálogos possíveis de utilização destacando-se os resultados obtidos naqueles que foram experimentados na UFRGS.

2. FATORES A CONSIDERAR NA ESCOLHA DE UM DIÁLOGO

A seleção de um diálogo apropriado sempre deve ser orientada ao atendimento das necessidades do usuário. Deve-se buscar na descrição das tarefas do usuário que serão automatizadas pelo uso do sistema interativo determinar suas necessidades de informações (dados buscados, forma de seleção e/ou agrupamento destes etc). As motivações, habilidades e treinamento passível de ser oferecido aos usuários dentro de um critério de mínimo custo - máximo benefício, também vão orientar a escolha desta ou daquela modalidade de diálogo.

Se a aplicação envolver atualização em tempo real dos dados o diálogo deverá conter uma preocupação muito maior com detecção dos erros que possam ser cometidos pelo usuário na informação dos dados a serem atualizados.

Quando o operador do terminal é um profissional especializado pode-se esperar que seja capaz de interagir numa linguagem baseada em abreviaturas ou codificações ou usando o seu jargão profissional. Se o usuário não for especialista mas operar regularmente o terminal com uma freqüência diária razoável, (mais de 10) pode-se também esperar que ele seja capaz de operar utilizando as abreviaturas ou codificações se o número de tais códigos ou mnemônicos não for excessivamente alto (máximo de 20). Mesmo para um operador regular, se obrigado a utilizar um número excessivamente alto de abreviaturas ou codificações, pode-se esperar que somente memorizará as freqüentemente utilizadas. Quando necessitar daquelas que são menos regularmente utilizadas, terá dificuldades, cometerá erros, e precisará de auxílio através de manuais ou mesmo de instrução on-line. Quando os usuários, além de não especialistas também são causuais, como em sistemas de ensino ou avaliação por terminal, o nível de codificação ou formatação que se impõe às mensagens deve ser o mínimo, pois não há possibilidade de treinar seja pelo ensino dos comandos seja pelo uso, operador para que dialogue com o sistema adequadamente.

Quanto às características dos terminais utilizados na interação homem-máquina, pode-se destacar o tipo de terminal (teleimpressor ou vídeo) como determinante da forma de diálogo. Quando o terminal é de vídeo, certas características, tais como campo protegido, destaques na tela e velocidade de transmissão também tem impacto na seleção de uma forma de diálogo a ser implementada. Um diálogo que exija transmissão de um grande vo

lume de caracteres com um terminal de baixa velocidade provocará um tempo demasiado alto de preenchimento da tela o que ocasionará desconforto para o usuário. Quando o terminal é inteligente pode-se programá-lo para efetuar algum tipo de validação localmente.

Poderíamos citar uma série de fatores que identificam um bom diálogo:

a) Fácil para aprender; o tempo de treinamento e necessidade de retreinamento não deve ser execessivo.

b) Fácil para usar: o usuário não deve ser obrigado a digitar grande número de teclas de controle e outros caracteres que não sejam absolutamente necessários na operação do terminal. Por outro lado espera-se que o usuário somente seja obrigado a dialogar usando termos que lhe são familiares.

c) Facilidade para adaptação e modificação; espera-se que a introdução de novas consultas ou novos itens em uma consulta não impliquem em reprogramação do sistema ou retreinamento excessivo.

d) Capacidade de detectar erros; os eventuais erros cometidos pelo usuário ao digitar devem ser facilmente detectados pelo sistema e o usuário deve poder corrigí-los de maneira clara e sucinta.

e) Eficiente; a quantidade de caracteres ou movimentos efetuados pelo usuário no terminal deve-se restringir ao mínimo evitando-se mensagens desnecessárias; somente dados relevantes devem ser integrantes do diálogo.

f) Consistente; as diferentes consultas que integram o diálogo devem apresentar uma certa analogia de aspecto, ordem nos itens apresentados ou solicitados e as diferentes teclas utilizadas em um terminal para fim de controle devem ser as mesmas nas diversas formas de consultas de um sistema.

g) Tutorial: quando o usuário comete erros o diálogo deve ter estabelecido uma previsão para, de uma maneira instrutiva, conduzí-lo ao modo certo de efetuar a consulta.

Espera-se que o diálogo, especialmente no caso usuários não especialistas, ou causuais ostente um baixo grau de codificação ou uso de abreviaturas e interaja com o operador do terminal em uma forma mais natural a este.

3. MODALIDADES DE DIÁLOGOS

Levando em consideração os fatores apontados na escolha de um diálogo, determina-se detalhes do mesmo. Existe uma ampla gama de possibilidades, as mais significativas das quais serão discutidas brevemente a seguir:

A) DIÁLOGO BASEADO EM LINGUAGEM NATURAL

Seria a forma ideal de diálogo, pois permitiria ao usuário expressar-se de modo mais flexível. Porém, na prática, apresenta muitas dificuldades; existe muita ambiguidade numa linguagem natural e dificuldades de nível sintático e semântico tornam a programação extremanente complexa.

Existem sistemas que utilizam apenas a análise sintática para manter conversações com o usuário - neste caso, a questão do usuário não é "entendida" mas apenas são aproveitadas certas palavras desta pergunta para que seja gerada uma resposta.

Num nível bem mais complexo, estão os sistemas interativos que "entendem" uma sentença do usuário. Isto é conseguido construindo-se uma representação do sentido da sentença, ou então fazendo inferências a respeito de fatos contidos num modelo geral.

O uso deste tipo de diálogo é acessível a qualquer usuário, mesmo sem treinamento.

B) DIÁLOGO BASEADO EM LINGUAGEM SEMI-NATURAL

Este tipo de diálogo permite o uso de um número restrito de palavras do vocabulário natural e de palavras-chaves pré-definidas.

A vantagem reside no fato de o usuário poder empregar palavras familiares.

Porém, o usuário corre o risco de superestimar a inteligência da máquina e usar vocabulário mais complexo do que o permitido.

Quando foi desenvolvido o sistema de recuperação de informações bibliográficas denominado SIRI (exemplos no anexo) observou-se exatamente esta característica uma vez que era comum os usuários cometerem erros por tentarem dar ordens ao computador utilizando verbos ou palavras chaves que o mesmo não conhecia. Nestes casos o sistema interagia solicitando um sinônimo para a palavra desconhecida e repetia-se esta operação até que tivesse reconhecido e identificado todos os vocábulos constantes no comando dado pelo usuário no terminal.

C) LINGUAGEM DE PROGRAMAÇÃO

As linguagens de programação, em geral, são para usuários treinados.

Porém existem linguagens mais simples, como BASIC , APL ou MUMPS, para que usuários não programadores possam desenvolver seus próprios programas on-line.

Este tipo de diálogo não é adequado a usuários casuais, pois exigem o conhecimento da linguagem utilizada. Na UFRGS, a experiência de utilização da linguagem MUMPS por um grupo de médicos que desenvolve seus próprios programas para recolher e manipular dados clínicos tratando-os estatisticamente, evidenciou, amplo sucesso, pois a aceitação da linguagem devido a facilidade de aprendizagem foi muito grande.

D) DIÁLOGO COM MNEMÔNICOS

Se por um lado, este tipo de diálogo facilita a análise pelo computador, por outro lado, exige que o usuário decore os mnemônicos existentes. Portanto a dificuldade aumenta se é grande o número de mnemônicos ou se o usuário é casual.

Um recurso útil é a apresentação da lista de mnemônicos com seus significados pelo terminal, quando o operador necessite.

E) DIÁLOGO ATRAVÉS DE PREENCHIMENTO DE FORMULÁRIO

Esta forma de diálogo é utilizada especialmente em terminais de vídeo, e é relativamente, fácil de usar.

Na tela é projetado um formulário ou mapa, com campos em branco, os quais deverão ser preenchidos pelo operador.

Este mapa é protegido, isto é, o usuário só poderá colocar dados em certas áreas pré-definidas. Ao pressionar-se a tecla de "ENVIO" somente os dados destes campos variáveis é que são transmitidos.

Para que seja possível este tipo de diálogo, o terminal deve ser um vídeo com capacidade de operar com campo protegido.

F) DIÁLOGO BASEADO EM ESCOLHA SIMPLES(OU SELEÇÃO EM MENU)

A grande vantagem da escolha simples é que se presta a qualquer tipo de usuário pois não exige aprendizagem de mnemônicos ou comandos.

O computador instrui sobre o que fazer a cada etapa, e o usuário só precisa selecionar uma das opções apresentadas.

Esta forma de diálogo é util para operadores casuais, pois necessita pouco ou nenhum treinamento. Porém, em algumas experiências com operadores regulares, observou-se uma certa rejeição a esta técnica: a simplicidade das respostas pode torná-la enfadonha.

Esta técnica foi utilizada conforme se pode ver pelo anexo no sistema de ensino por terminal na UFRGS, que é oferecido a aluno sem qualquer treinamento prévio no manejo de terminais ou de alguma linguagem de programação.

G) DIÁLOGO BASEADO EM RESPOSTA CURTA (OU INSTRUÇÃO E RESPOSTA)

O computador formula perguntas ao usuário, já induzindo as respostas possíveis, e instruindo como responder. O usuário deverá escolher uma das alternativas e digitá-la (ou digitar apenas parte dela).

Esta forma de diálogo é recomendada para usuários pouco freqüentes (ver exemplo no sistema traçador interativo no anexo).

Sua desvantagem encontra-se no fato de os formatos de entrada serem ainda muito rígidos.

H) DIÁLOGO BASEADO EM COMANDOS DE PROGRAMA

Consiste no uso de uma linguagem de programação orientada à solução de um tipo especial de problema.

Sua desvantagem reside no fato de não ter a flexibilidade de uma linguagem de programação comum.

Pode ser implementada através da chamada de rotinas e exige treinamento para sua utilização.

I) DIÁLOGO COM FORMATO EM DISPLAY

É um estilo muito simples de diálogo: o sistema pede os dados que necessita e já dá o formato como o operador deve digitá-los (ex.: separados por "/" ou vírgula, etc).

É eficiente em grande número de aplicações e na maioria dos terminais, principalmente com usuários casuais.

No caso de usuários experientes, pode-se oferecer a opção de omitir os formatos na tela.

J) DIÁLOGO COM MODIFICAÇÃO DE PAINEL

Esta forma de diálogo é usado principalmente para administração de banco de dados, sendo recomendada apenas a operadores experientes.

O operador apresenta uma chave de entrada, obtendo na tela a resposta. Pode, então modificar alguns dos campos que necessitem atualização, e enviá-los de volta ao sistema.

Para maior segurança, recomenda-se que ao ser usado este tipo de diálogo, restrinja-se aos campos modificáveis e que seja transferido para um arquivo histórico os valores atualizados para posterior conferência ou restauração em caso de necessidade.

K) DIÁLOGOS HÍBRIDOS

Na maioria dos casos, o diálogo é inteiramente iniciado ou pelo operador ou pelo computador.

Os diálogos híbridos são uma mistura destes dois casos. São projetados para usuários especializados numa certa área de conhecimento e requerem treinamento. O usuário apresenta ordens ao sistema, sendo interpretado passo a passo. Nos casos em que o usuário não informa todos os dados necessários ou informa algum incorretamente, o sistema solicita a correção apenas do item omisso ou incorreto.

L) HARDWARE ESPECIAL

Pode-se utilizar um terminal especialmente projetado para uma determinada aplicação, com chaves e luzes rotuladas com os elementos do diálogo, ou ainda leitores de cartões ou de marcas óticas.

Há desvantagens, neste caso, devido à falta de flexibilidade, no caso de serem necessárias algumas alterações no diálogo, mas existe a vantagem de minimizar os movimentos do operador e diminuir o custo do terminal colocando nele apenas as teclas e luzes necessárias à aplicação.

M) DIÁLOGO COM OPERADOR TOTALMENTE DESTREINADO

Neste tipo de diálogo, deve-se tomar o cuidado de impedir que o operador entre com dados que o computador não possa interpretar.

A maneira de tornar isto possível é restringir as teclas que o operador pode usar (por exemplo, só três teclas disponíveis: SIM, NÃO e NÃO SEI), e ainda projetar diálogos sempre iniciados pelo computador (por exemplo, com escolha simples).

É importante que as mensagens sejam simples e claras.

N) LINGUAGEM ESPECÍFICA PARA CONSULTA A BANCO DE DADOS

Esta forma de diálogo é voltada a usuários que desejam fazer pedidos ou pesquisas em banco de dados.

Inicialmente o usuário indica o conjunto de dados que deseja pesquisar. O sistema mostra no display os elementos do registro relacionado. O usuário então entra com os parâmetros fixos de pesquisa, e a variável desconhecida que deseja, recebendo a resposta ou tabela pedida.

4. CONSIDERAÇÕES DIVERSAS

4.1. FORMATO DE MENSAGENS

As mensagens, num diálogo podem ter formato livre ou formato fixo.

Quando as mensagens tem formato livre, deve-se tomar os seguintes cuidados:

a) usar delimitadores com fácil localização no teclado
b) não confundir delimitador entre itens com outros existentes (por exemplo, em datas)
c) em caso de itens com limite de tamanho, usar um "prompt" ou guia para o usuário
d) projetar os formatos para que sejam acessíveis a não-datilógrafos
e) não pedir confirmações desnecessárias
f) se for necessário, reapresentar a mensagem que entrou

No caso de mensagens com formato fixo, é aconselhavel cuidados com seqüenciação lógica, espacejamento, relevância, consistência, agrupamento e simplicidade da mensagem. Neste caso é desaconselhável o uso de abreviaturas ou codificações desnecessárias; o uso de jargão do computador e não do usuário; a formatação pouco clara do item; e o uso em excesso de "atrativos" de atenção (por exemplo: sublinhados, etc).

4.2. CODIFICAÇÃO (SUBSTITUIÇÃO DE TERMOS POR SÍMBOLOS)

Observou-se que o aumento da codificação pode aumentar o throughput, mas causa uma diminuição na legibilidade e na facilidade de aprendizagem e um aumento do número de erros.

As técnicas de codificação mais usadas são:

a) Por cópia: parte do dado a ser referenciado é copiada diretamente.
   ex.: PRINT - P
b) Associativa: o item é associado a um valor particular
   ex.: VERMELHO = 1
        AZUL = 2
c) Transformacional: há um conjunto de regras para derivar o código
   ex.: "ENTRE COM A INICIAL DE CADA PALAVRA".

Os erros de codificação em geral são devidos a apenas um caracter errado - assim, isto pode ser evitado, ou reduzido, pelo uso de redundância.

Quando mais ocasional for o usuário, menos codificação deve ser usada. Para facilitar uma entrada ou a visualização de uma resposta pode-se agrupar itens muito longos (numéricos).

Além disso, pode-se também buscar o auxílio do computador para expandir códigos curtos.

4.3. MENSAGENS DE ERROS

É recomendável que as mensagens de erros sejam auto-explicativas e que se evite ao máximo a codificação. As mensagens são especificadas pelo projetista mas quem realmente testa sua eficiência é o usuário. Elas devem ser adaptáveis à experiência do usuário e à regularidade de uso.

5. ETAPAS NO PROJETO DE UM DIÁLOGO

As diferentes formas de diálogo mostradas, são possíveis alternativas que devem ser analisadas com vistas a determinar-se a forma mais adequada ao sistema particular em questão. Contudo, dificilmente se encontrará um exemplo ou modelo pronto para ser usado. Na maioria das vezes será necessário todo um estudo para determinar as modalidades mais adequadas e as adaptações que se farão necessárias.

No desenvolvimento do projeto de um diálogo, pode-se estabelecer cinco etapas e, em cada uma, atinge-se um grau maior de detalhamento, numa abordagem "top-down". A interdependência das etapas pode ser a da figura 2.

AS ATIVIDADES PERTINENTES A CADA ETAPA SÃO AS SEGUINTES:

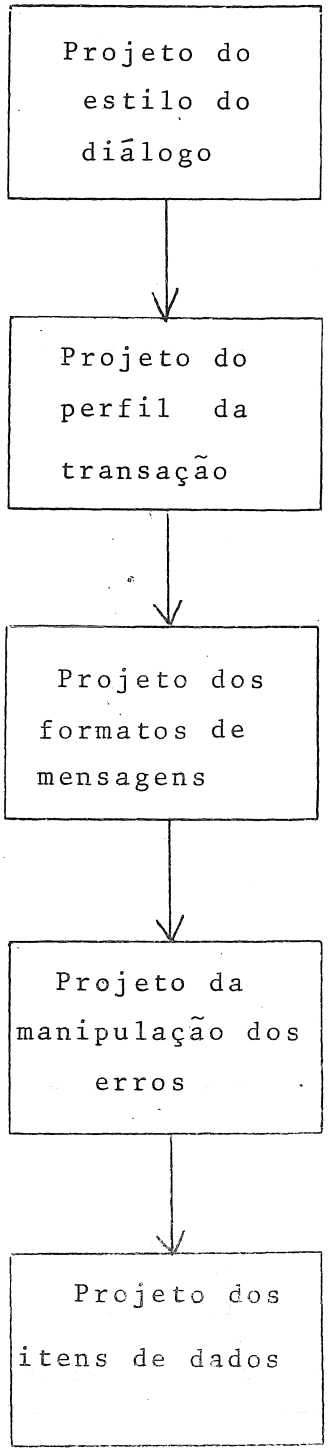
a) Projeto do estilo do diálogo: Esta etapa consiste da identificação dos fatores que tem influência na aplicação, e da seleção e combinação das escolhas mais adequadas.

b) Projeto do perfil da transação: consiste da investigação das exigiências do usuário e da aplicação, dos possíveis perfis da transação e da obtenção da opinião do usuário a este respeito.

c) Projeto dos formatos de mensagens: nesta etapa deve-se tratar do layout e codificação, visando obter eficiência na entrada de dados e clareza na saída dos resultados.

d) Projeto da manipulação dos erros: nesta etapa deve-se planejar os métodos que serão utilizados para evitar, detectar e corrigir erros.

FIGURA 2: ETAPAS NO PROJETO DE UM DIÁLOGO



e) Projeto dos itens de dados: refere-se ao uso de codificação, problemas de delimitadores, etc.

6) CONCLUSÕES

Criar um diálogo homem/máquina pode ser mesmo considerado uma obra de arte. Existe uma tecnologia já estabelecida a respeito, mas é realmente implantando e observando os resultados desta ou daquela forma de diálogo que se descobre as sutilezas que diferenciam um bom de um mau diálogo.

A disposição das informações em um vídeo exige uma certa estética e frequentemente, se incorre em um erro que é o de tentar projetar na tela a maior quantidade de informações que esta comporte. Isto em primeiro lugar torna difícil e monótona a leitura da mesma e em caso de haver realmente necessidade de inserir mais algum campo não haverá lugar para este. Este é apenas um exemplo do que acontece quando se realmente vê na tela o diálogo que se projetou no papel. Descobre-se que a disposição dos campos não está adequada, que não se destacou suficientemente determinada informação etc. Por isso, reenfatiza-se ser a experiência a melhor maneira de aprimorar a técnica para projetar e implantar diálogos adequados ao contexto em que serão utilizados.

BIBLIOGRAFIA

MARTIN, James. Systems analysis for data transmission. New Jersey, Prentice-Hall, Inc., 1972.

MARTIN, James. Telecomunications and the computer. New Jersey, Prentice-Hall, Inc., 1969.

Man/Computer Communication. Infotech-State of the art - Report Maidenhead (England), Infotech International. 1979.

D'AZEVEDO, M.C. Comunicação, linguagem, automação. Porto Alegre, Comissão Central de Publicaçoes UFRGS, 1978.

TAROUCO, Liane M.R. & SABANI, Cláudia. Metodologia para desenvolvimento de sistemas on-line, Porto Alegre, 1979.

Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

TRANSFORMACION DE FIGURAS Y GENERACION DE FIGURAS INTERMEDIAS
UNA APROXIMACION A LA ANIMACION

Mario Colosso V.
Universidad Simón Bolívar
Caracas, Venezuela

Introducción

Teniendo una figura inicial y otra final en un mismo plano, cómo generar una secuencia de figuras intermedias que lleven de la figura inicial a la figura final en forma casi imperceptible? Esta pregunta es la que suscita el desarrollo del presente trabajo.

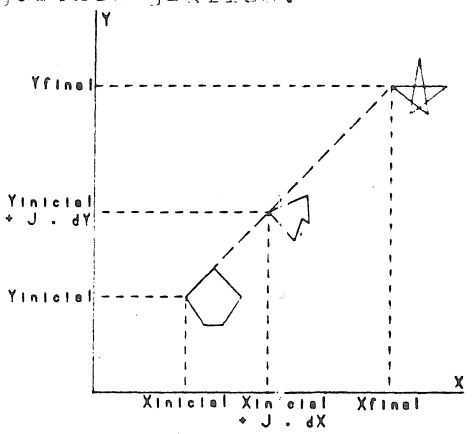
Trataremos de dar aquí una idea del desarrollo de los conceptos empleados, así como sugerencias para su posterior evolución.

Desarrollo

Una figura no es mas que una secuencia de puntos unidos entre sí, segun una cierta ley de conectividad. Segun esta definición, el análisis del problema propuesto puede ser

llevado al análisis de los puntos que conforman las figuras.

Tengamos la siguiente gráfica:



donde el punto $(X_{inicial}, Y_{inicial})$ de la figura inicial será transformado en el punto (X_{final}, Y_{final}) de la figura final. La transformación del punto $(X_{inicial}, Y_{inicial})$ en su correspondiente, en la j-ésima figura intermedia, esta dada por

$$X = X_{inicial} + j \cdot d_x$$
$$Y = Y_{inicial} + j \cdot d_y$$

Los términos d_x y d_y corresponden a la longitud de los intervalos en X y Y.

Para un total de N figuras igualmente espaciadas a ser generadas,

$$d_x = (X_{final} - X_{inicial}) / N$$
$$d_y = (Y_{final} - Y_{inicial}) / N$$

entonces, las coordenadas del punto (X, Y) en la j-ésima figura intermedia, estan dadas por:

$$X' = X + (j / N) \cdot (X_{final} - X_{inicial})$$
$$Y' = Y + (j / N) \cdot (Y_{final} - Y_{inicial})$$

Vemos la aparición del término (j / N), con valores comprendidos en el rango [0,1], el cual podemos interpretar como un "porcentaje de transformación" de una figura a otra. Así, si el porcentaje de transformación es igual a 0, tenemos la figura inicial, y si es igual a 1, la figura final.

Entonces, una primera aproximación al algoritmo buscado, para el caso de una figura conexa, es la siguiente:

Sea INC = 1 / (Número de figuras - 1)

Variando el porcentaje de trasformación, P, entre 0 y 1, con incremento INC .

Para todos los puntos

$$X = X_{inicial}(j) + P \cdot (X_{final}(j) - X_{inicial}(j))$$
$$Y = Y_{inicial}(j) + P \cdot (Y_{final}(j) - Y_{inicial}(j))$$

Si es el primer punto, entonces

MOVER (X, Y)

sino

DIBUJAR (X, Y)

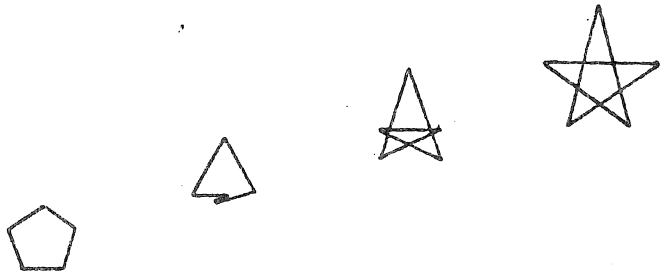


Figura 1.

Pero el algoritmo introduce un par de restricciones: la primera es que el número de puntos de la figura inicial debe ser igual al número de puntos de la figura final, y la segunda es que las figuras no pueden estar formadas por "objetos" disconexos; ésto es, cómo indicar que en un instante determinado queremos que el trazo sea invisible en lugar de visible (MOVER en lugar de DIBUJAR)?. Esta segunda es una restricción importante, limitando la cantidad de figuras que pueden ser manejadas. Debido a ello, dirigiremos nuestra atención hacia este punto.

Hemos caracterizado cada punto mediante dos atributos: las coordenadas X y Y que indican su posición en el plano. Agreguemos un tercer atributo por punto, el "atributo de visualización", con valores discretos 0 ó 1 y con el siguiente sentido:

> Si el atributo de visualización toma el valor 1, el segmento entre el punto anterior y el actual es visible, y si toma el valor 0, es invisible.

Indiquemos por V el atributo de visualización.

El atributo de visualización del j-ésimo punto de una figura intermedia esta dado por:

$$V = V_{inicial}(j) + P \cdot (V_{final}(j) - V_{inicial}(j))$$

y tiene valores en el rango [0,1]. (Notemos la similitud con las ecuaciones de transformación definidas anteriormente.) Dado

que los dispositivos gráficos disponibles no poseen la característica de "tonos grises" o "medias-tintas", el atributo de visualización, V, debe ser llevado a valores discretos 0 ó 1.

Una primera aproximación es la de tener:

V' = Parte entera (V + 0.5)

donde el factor de redondeo 0.5 puede ser pensado como un "porcentaje de corte". En este caso indica que, para aquellos puntos que deban invertir su atributo de visualización, a la mitad de la transformación los segmentos visibles se transforman en invisibles, y viceversa.

Una generalización es la de tener el porcentaje de corte variable, permitiendo modificar el momento de la transición.

Veamos esta segunda aproximación del algoritmo:

Sea INC = 1 / (Número de figuras - 1)

Variando el porcentaje de transformación, P, entre 0 y 1, con incremento INC

Para todos los puntos

$$X = X_{inicial}(j) + P \cdot (X_{final}(j) - X_{inicial}(j))$$
$$Y = Y_{inicial}(j) + P \cdot (Y_{final}(j) - Y_{inicial}(j))$$
$$V = V_{inicial}(j) + P \cdot (V_{final}(j) - V_{inicial}(j))$$

V' = Parte entera (V + Porcentaje de corte)

TRAZAR (X, Y, V')

donde el procedimiento TRAZAR se define como:

```
TRAZAR (X, Y, V)

    Si V = 0, entonces
        MOVER (X, Y)
    sino
        DIBUJAR (X, Y)
```

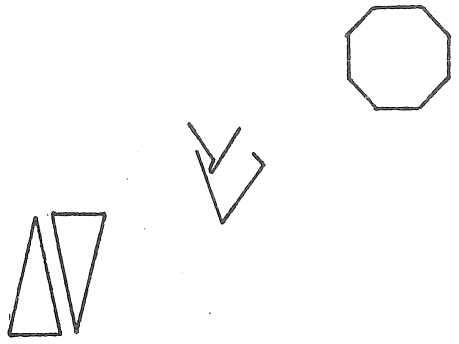


Figura 2.

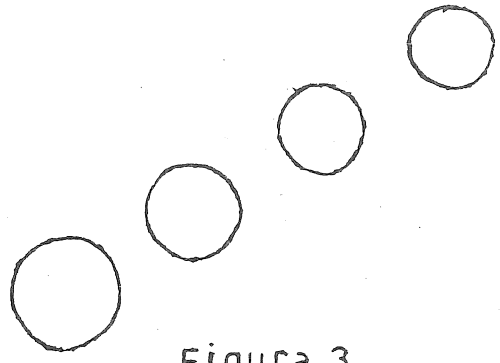


Figura 3.

Hemos, pues, alcanzado el primer objetivo: la generación de figuras intermedias. Sin embargo, las figuras generadas distan mucho del objetivo general propuesto: una aproximación a la animación. Falta aún esta "chispa de vida" que las caracteriza.

Si observamos la figura 3, notaremos que si la traslación se hubiese realizado a través de una curva en lugar de una recta, mejoraría la imagen. Generalizando, sería deseable poder trasladar las figuras segun cualquier curva, poder rotarlas a medida que se trasladan, transfomar una figura en otra a velocidad variable, etc.

Gracias a las sugerencias del profesor Bernardo Paris -figuras centradas en el origen y traslación del centro geo-

métrico de las figuras rotadas- fue posible la continuación de este trabajo.

Debemos, entonces, expresar cada punto en función del centro geométrico de la figura:

$$CGX = X_{mínimo} + (X_{máximo} - X_{mínimo}) / 2$$
$$CGY = Y_{mínimo} + (Y_{máximo} - Y_{mínimo}) / 2$$

son las coordenadas del centro geométrico de la figura, y

$$X' = X - (X_{mínimo} + (X_{máximo} - X_{mínimo}) / 2)$$
$$Y' = Y - (Y_{mínimo} + (Y_{máximo} - Y_{mínimo}) / 2)$$

son las coordenadas de un punto en función del centro geométrico. Estos nuevos puntos nos dan la figura ya trasladada al origen.

Podemos ahora rotar la figura y trasladarla a su posición correspondiente.

Las ecuaciones empleadas para rotar un punto un ángulo ɑ son:

$$X' = X \cdot \text{coseno}(ɑ) + Y \cdot \text{seno}(ɑ)$$
$$Y' = Y \cdot \text{coseno}(ɑ) - X \cdot \text{seno}(ɑ)$$

y darlo son:

$$X' = X + \text{Traslación en X}$$
$$Y' = Y + \text{Traslación en Y}$$

Uniendo ambas ecuaciones:

$$X'' = X' \cdot \text{coseno}(\alpha) + Y' \cdot \text{seno}(\alpha) + \text{Traslación en X}$$

$$Y'' = Y' \cdot \text{coseno}(\alpha) - X' \cdot \text{seno}(\alpha) + \text{Traslación en Y}$$

donde

$$X' = X + \text{Porcentaje de transformación} \cdot (X_{final} - X_{inicial})$$

$$Y' = Y + \text{Porcentaje de transformación} \cdot (Y_{final} - Y_{inicial})$$

Dejemos por un momento lo relativo a las traslaciones y enfoquemos nuestra atención a la proposición de transformación a velocidad variable. Esto significa que el porcentaje de transformación no debe ser linealmente creciente, sino una función.

Sea VT esta función, tal que VT:[0,1]->[0,1]. Retornando valores cercanos a 0 se tiene la figura inicial con poca distorsión, y cercanos a 1, la figura final. Esto implica que si la función VT toca varias veces 0 y 1 en el rango, ésto hace que se visualicen varias veces las figuras inicial y final. Denotando por P el porcentaje de transformación:

$$X' = X + VT(P) \cdot (X_{final} - X_{inicial})$$

$$Y' = Y + VT(P) \cdot (Y_{final} - Y_{inicial})$$

Volvamos a las traslaciones.

Sea $(CGX_{inicial}, CGY_{inicial})$ el centro geométrico de la figura inicial, y $(CGX_{final}, CGY_{final})$ el centro geométrico de la figura final.

Definamos las traslaciones de la siguiente manera:

$$\text{Traslación en X} = CGX_{inicial} + TX(P) \cdot (CGX_{final} - CGX_{inicial})$$

$$\text{Traslación en Y} = CGY_{inicial} + TY(P) \cdot (CGY_{final} - CGY_{inicial})$$

donde TX, TY:[0,1]->[0,1] e indican porcentajes de traslación en X y Y, respectivamente. La forma de definición de las traslaciones las fuerzan a quedar enmarcadas en la ventana generada por los centros geométricos de las figuras inicial y final.

Si tenemos las siguientes funciones para TX y TY:

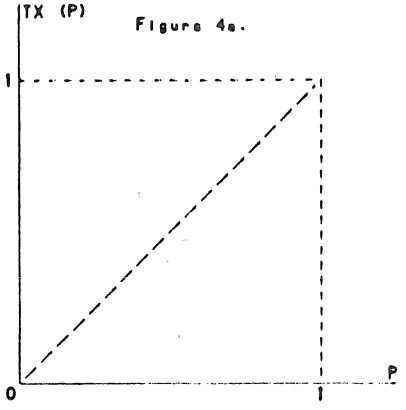


Figura 4a.

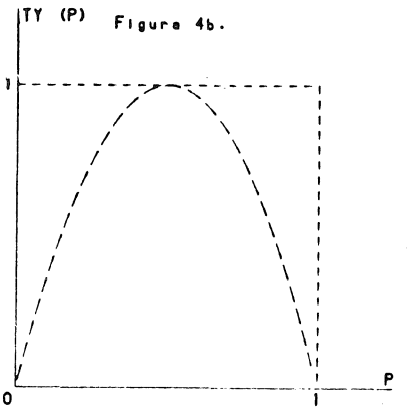


Figura 4b.

y si la disposición de los centros geométricos es como se muestra en 5a, tendremos la traslación del centro geométrico indicada en la figura 5b

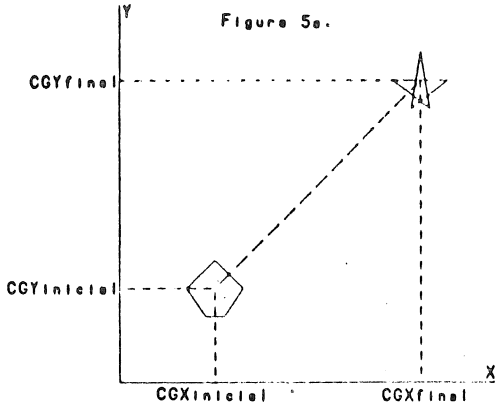


Figura 5a.

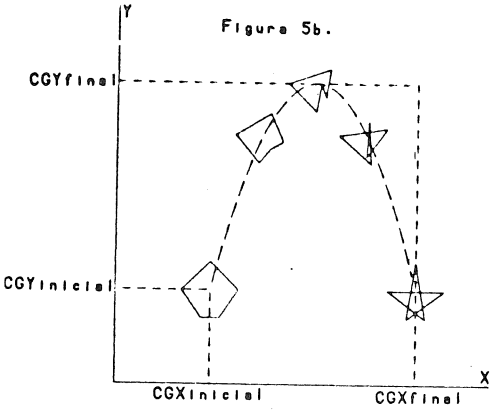


Figura 5b.

mientras que si la disposición es como en la figura 5a, el efecto obtenido es el mostrado en 5b

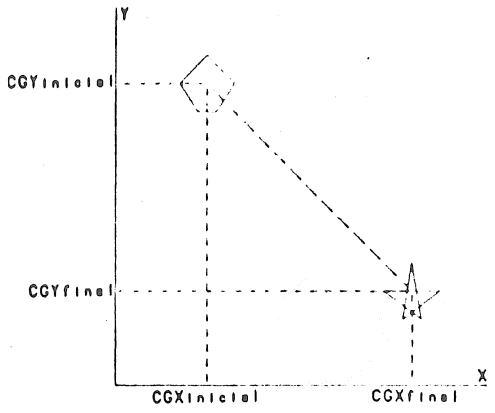


Figura 6a.

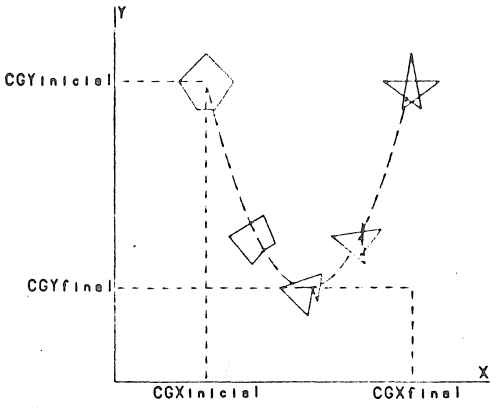


Figura 6b.

Unamos todos los conceptos previamente descritos:

Sean

INC = 1 / (Número de figuras - 1)

α = Angulo inicial

α_{inc} = (Angulo final - Angulo inicial) / (Número de figuras - 1)

Variando el porcentaje de transformación, P, entre 0 y 1, con incremento INC

Para todos los puntos

$$X = X_{inicial}(j) + VT(P) \cdot (X_{final}(j) - X_{inicial}(j))$$

$$Y = Y_{inicial}(j) + VT(P) \cdot (Y_{final}(j) - Y_{inicial}(j))$$

$$V = V_{inicial}(j) + VT(P) \cdot (V_{final}(j) - V_{inicial}(j))$$

$$X' = X \cdot \text{coseno}(\alpha) + Y \cdot \text{seno}(\alpha) + CGX_{inicial} + TX(P) \cdot (CGX_{final} - CGX_{inicial})$$

$$Y' = Y \cdot \text{coseno}(\alpha) - X \cdot \text{seno}(\alpha) + CGY_{inicial} + TY(P) \cdot (CGY_{final} - CGY_{inicial})$$

V' = Parte entera (V + Porcentaje de corte)

TRAZAR (X', Y', V')

$\alpha = \alpha + \alpha_{inc}$

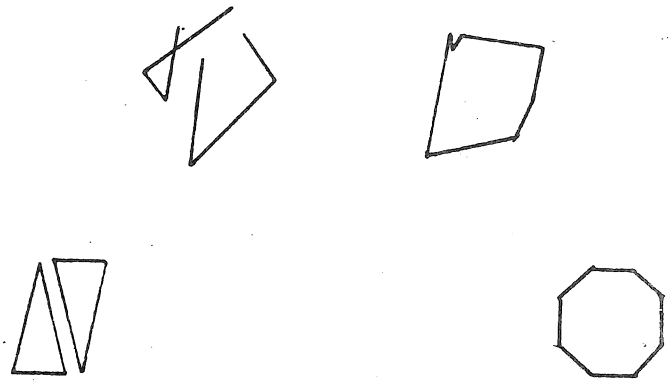


Figura 7.

Observando detenidamente los dos ejemplos siguientes notaremos que, seleccionando convenientemente las funciones de velocidad de transformación y traslaciones, podemos aproximar la generación a una situacion mas real.

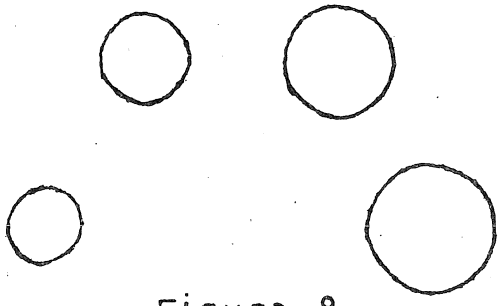


Figura 8.

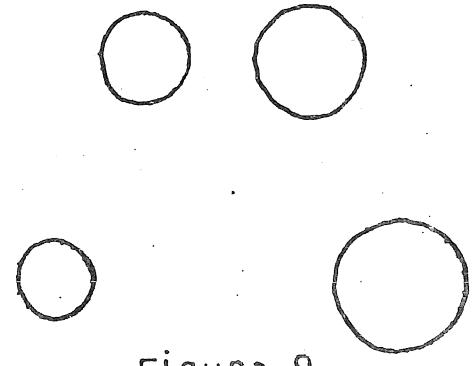


Figura 9.

Sin embargo, al tener el ángulo de rotación linealmente creciente, todavía no brindamos una flexibilidad total en cuanto a la movilidad de las figuras.

Definamos α de la siguiente manera:

$$\alpha = \text{Angulo inicial} + TA(P) \cdot (\text{Angulo final} - \text{Angulo inicial})$$

donde $TA:[0,1]\rightarrow[0,1]$ y representa un "porcentaje de rotación".

Casi sin darnos cuenta, todo el desarrollo se ha hecho en un plano; sin embargo, el espacio tridimensional ampliaría

grandemente las posibilidades de generación, implicando el empleo de algoritmos de proyección perspectiva cónica o equivalentes, para la visualización de las imagenes.

Enfoquemos ahora nuestra atención a la transformación del algoritmo anterior a un espacio de tres dimensiones.

Primeramente, cada punto debe estar caracterizado mediante cuatro atributos: las coordenadas X, Y y Z que indican su posición en el espacio, y el atributo de visualización.

Seguidamente, debemos definir las ecuaciones de rotación de un punto un ángulo α en el plano X-Y, y un ángulo β en el plano Y-Z:

$$X' = (X \cos(\alpha) + Y \sin(\alpha)) \cdot \cos(\beta) + Z \sin(\beta)$$

$$Y' = Y \cos(\alpha) - X \sin(\alpha)$$

$$Z' = Z \cos(\beta) - (X \cos(\alpha) + Y \sin(\alpha)) \cdot \sin(\beta)$$

Como hemos dicho anteriormente, los ángulos de rotación no deben ser linealmente crecientes, sino que deben ser expresados como funciones. Así

$$\alpha = \alpha_{inicial} + TA(P) \cdot (\alpha_{final} - \alpha_{inicial})$$

$$\beta = \beta_{inicial} + TB(P) \cdot (\beta_{final} - \beta_{inicial})$$

donde TA, $Tb:[0,1] \rightarrow [0,1]$ y representan porcentajes de rotación.

Finalmente, las figuras tridimensionales generadas deben ser

proyectadas en un plano para su posterior representación en el dispostivo. Escogimos para ello una proyección axonométrica desarrollada por B. Troncone (UCV), la cual presenta como característica fundamental el proveer mas de un "punto de fuga":

$$X' = Y \text{ coseno } (\alpha_{visual}) - X \text{ seno } (\alpha_{visual})$$

$$Y' = Z \text{ coseno } (\beta_{visual}) - X \text{ coseno } (\beta_{visual}) \text{ seno } (\alpha_{visual}) - Y \text{ seno } (\beta_{visual}) \text{ seno } (\alpha_{visual})$$

donde α_{visual} es el ángulo de rotación de la visual respecto al eje X, y β_{visual} es el ángulo de alzada de la visual respecto al plano X-Y. El origen del espacio tridimensional se proyecta en el plano como el punto de coordenadas (0, 0).

Veamos esta última aproximación al algoritmo buscado:

Sea INC = 1 / (Número de figuras - 1)

Variando el porcentaje de transformación, P, entre 0 y 1, con incremento INC

$$\alpha = \alpha_{inicial} + TA(P) \cdot (\alpha_{final} - \alpha_{inicial})$$

$$\beta = \beta_{inicial} + TB(P) \cdot (\beta_{final} - \beta_{inicial})$$

Para todos los puntos

$$X = X_{inicial}(j) + VT(P) \cdot (X_{final}(j) - X_{inicial}(j))$$

$$Y = Y_{inicial}(j) + VT(P) \cdot (Y_{final}(j) - Y_{inicial}(j))$$

$$Z = Z_{inicial}(j) + VT(P) \cdot (Z_{final}(j) - Z_{inicial}(j))$$

$$V = V_{inicial}(j) + VT(P) \cdot (V_{final}(j) - V_{inicial}(j))$$

$$X' = (X \cos(\alpha) + Y \operatorname{sen}(\alpha)) \cdot \cos(\beta) + Z \operatorname{sen}(\beta) + CGX_{inicial} + TX(P) \cdot (CGX_{final} - CGX_{inicial})$$

$$Y' = Y \cos(\alpha) - X \operatorname{sen}(\alpha) + CGY_{inicial} + TY(P) \cdot (CGY_{final} - CGY_{inicial})$$

$$Z' = Z \cos(\beta) - (X \cos(\alpha) + Y \operatorname{sen}(\alpha) \cdot \operatorname{sen}(\beta) + CGY_{inicial} + TY(P) \cdot (CGY_{final} - CGY_{inicial})$$

$$X'' = Y' \cos(\alpha_{visual}) - X' \operatorname{sen}(\alpha_{visual})$$

$$Y'' = Z' \cos(\beta_{visual}) - X' \cos(\beta_{visual}) \operatorname{sen}(\alpha_{visual}) - Y' \operatorname{sen}(\beta_{visual}) \operatorname{sen}(\alpha_{visual})$$

V' = Parte entera (V + Porcentaje de corte)

TRAZAR (X", Y", V')

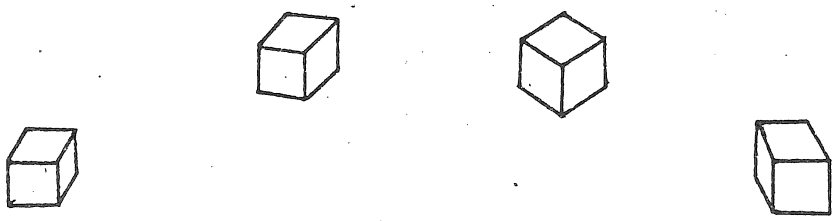


Figura 10.

Evolución

Hemos descrito el desarrollo de este trabajo hasta el momento actual. Pero, qué tendencias o generalizaciones sería conveniente introducir?

Una primera tendencia sería la de tratar de eliminar la restricción de igualdad de número de puntos de las figuras inicial y final.

Ahora bien, por qué tener sólo una figura inicial y una figura final, y no tener varias intermedias adicionales?. Esto es, el poder definir escenas completas a través de la especificación de algunas figuras claves. Ello causaría una animación más dinámica.

Implementación

La implementación del algoritmo previamente descrito fue realizada inicialmente en el lenguaje Pascal, disponible en un DECsystem-10 bajo TOPS-10, y posteriormente en el Lenguaje C en una PDP-11/45 bajo UNIX, ambas en la Universidad Simón Bolívar.

Como característica resaltante de la misma, permite la generación de las figuras en los dispositivos Plotter CalComp 1037, Tektronix 4006 y Hewlett Packard 2648A, disponibles en el Laboratorio de Computación Gráfica.

INDICE TOMO 1

CAPITULO C - BASE DE DATOS

CAPITULO D - HARDWARE

Este libro se terminó de imprimir
el 27 de marzo de 1981, en
Artes Gráficas B. U. Chiesino S.A.
Ameghino 838 - Avellaneda - Bs. As.

Esta edición es de 1.500 ejemplares.