CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL IPN

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

PROGRAMACION AUTOMATICA A PARTIR DE DESCRIPTORES DE FLUJO DE INFORMACION

TESIS QUE PARA OBTENER EL GRADO DE

DOCTOR EN CIENCIAS PRESENTA EL M. en C.

SERGIO VICTOR CHAPA VERGARA

TRABAJO DIRIGIDO POR LA DOCTORA

ANA MARIA MARTINEZ ENRIQUEZ

México, D. F. Marzo de 1991.

XD
91.
09
00.
5 91

A mis padres
Alfredo y Ana Maria.

A mi esposa Karla
e hijo Sergio Alfredo

A mi maestro
Dr. Harold V. McIntosh

AGRADECIMIENTOS

Primero quiero agradecer a las Instituciones que de alguna forma me apoyaron en la realización de mi doctorado. Al Consejo NAcional de Ciencia y Tecnología (CONACyT), quiero agradecer el apoyo de beca que recibí durante el tiempo que tomé los cursos de doctorado. Al Sistema Nacional de Investigadores agradezco su apoyo económico para un mejor desempeño de mis actividades. Y al Centro de Investigación y de Estudios Avanzados por las facilidades prestadas para el desarrollo del proyecto.

A la Doctora Ana María Martínez E., quiero agradecer su dirección a este trabajo y las revisiones al mismo. Al Dr. Adolfo Guzmán Arenas, quién fué la persona que propuso inicialmente el tema, quiero dar un especial reconocimiento por su crítica valiosa y la revisión meticulósa que hizo al documento. Al Dr. José Luis Gordillo M., Dr. Guillermo Morales L. y Jaime Rangel M., les agradezco sinceramente sus comentarios que me permitierón mejorar el documento.

Es el momento adecuado para agradecer a todos mis maestros de licenciatura, maestría y doctorado; su contribución a mi formación profesional; considerando en forma especial al Dr. Harold V. McIntosh.

También, quiero agradecer a mis alumnos de licenciatura de la ENEP-Acatlán, la atención que tuvieron a mis cursos de base de datos soportando temas como: descriptor de archivos, intérprete ISBL y herramienta de "software"; permitiendome de esta forma tener interlocutores para madurar el proyecto. En particular, al Sr. Rubén Gonzalez Ramírez y Sr. Héctor Sánchez Pérez por sus ayudas en programación un sincero agradecimiento.

A mi familia quiero agradecer el apoyo que siempre he tenido de ellos, especialmente a mis padres y esposa Karla que en momentos difíciles me apoyaron moralmente. A mis amigos agradezco sus atenciones y de forma especial al M. en C. Manuel Gonzalez Hernández, agradezco su apoyo que siempre me ha brindado.

I N D I C E

LISTA DE FIGURAS

CAPITULO 4

CAPITULO 5

CAPITULO 6

INTRODUCCION

Este trabajo tiene por objetivo el de presentar un nuevo lenguaje de tipo visual y el sistema que lo sustenta para poder desarrollar aplicaciones. Por medio de este lenguaje iconográfico, es posible plantear problemas con diagramas denominados flujogramas, resolviendose automáticamente mediante la generación de programas dentro del área de procesamiento de datos. Con el objetivo de evaluar el lenguaje en este trabajo doy un marco teórico acerca de la metodología de Programación Automática (PA) y la naturaleza de los lenguajes visuales.

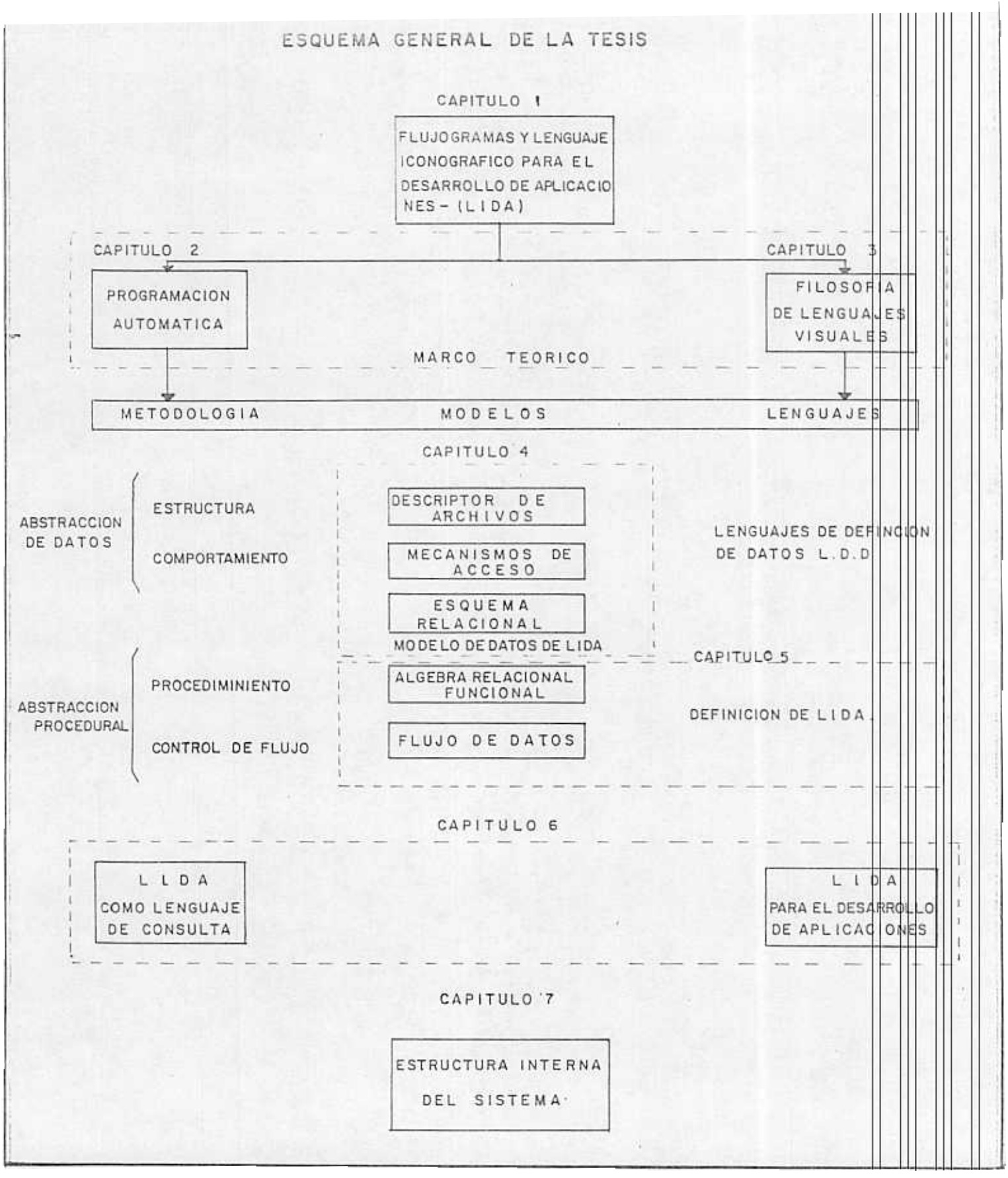
Acerca de los términos "Programación Automática" y "Descriptores del Flujo de Información" que constituyen el título de la tesis, son los que se conjugan para desarrollar un nuevo Lenguaje de Muy Alto Nivel (LMAN) de características gráficas denominado "Lenguaje Iconográfico para el Desarrollo de Aplicaciones" LIDA.

Por un lado, el término Descriptores de Flujo de Información debe entenderse como un medio de expresión gráfica descrito con diagramas de flujo de datos (flujogramas) para plantear problemas. Por otro lado, con PA se sintetizan programas automáticamente para la solución de los problemas descritos en una carta de flujograma. El objetivo de LIDA es el de tener un nivel de representación visual con diagramas y descripciones icónicas; y la generación automática de código para la solución de los problemas planteados.

En esta introducción presento un diagrama de la estructura general de la tesis en donde se sintetizan las diferentes partes describiendo las interrelaciones de las mismas. Dos principales son los que intervienen: Programación Automática y los Lenguajes Visuales. Del primero se desprende las metodologías de abstracción procedural y de datos; siendo estas dos la base para la construcción de los LMAN. Acerca de los Lenguajes Visuales en la tesis dedico un sustancial espacio para definir la naturaleza y la contribución de varios de ellos.

El capitulo 1 tiene por objetivo dar el planteamiento del concepto de flujograma para describir poco después que es LIDA. En este mismo capitulo se dá la estructura general del sistema y cómo esta constituido en sus partes diversas, presentando el procedimiento de manejo de cada una de ellas en una sesión de trabajo. Debido a que en el trascurso del desarrollo del trabajo algunas etapas han originado resultados y publicaciones, he creído conveniente dar una reseña breve del trabajo, mostrando como los diferentes módulos se desarrollaron para ensamblarse posteriormente.

ESQUEMA GENERAL DE LA TESIS



Programación Automática como parte constitutiva del título es uno de los principales elementos de la tesis; razón por la cual, el capítulo 2 la trata en tres subtemas fundamentales: programación automática e inteligencia artificial; metodología en los lenguajes de muy alto nivel; y una clasificación de los mismos. El primer tema trata de como el concepto alcanza la esfera de la inteligencia artificial con la idea de construir y ejecutar automáticamente programas. El segundo trata con la abstracción procedural y abstracción de datos, que caracterizan a los LMAN. Para terminar el capítulo doy el marco teórico de los LMAN con una clasificación de los lenguajes: de propósitos generales, para la inteligencia artificial y el procesamiento de datos.

Debido a qué, se quiere mostrar la contribución de LIDA dentro del ámbito de los lenguajes de tipo visual. El objetivo del capítulo 3 es el de dar la filosofía y naturaleza de los lenguajes visuales; con una evolución que muestra las contribuciones de lenguajes en diferentes contextos. La idea es presentar cómo las imágenes gráficas proporcionan un gran recurso para la programación, siendo los diagramas los paradigmas para la creación de lenguajes visuales.

En este capítulo 3 doy una discusión del uso amplio de los lenguajes gráficos en ingeniería de "software" para el análisis y el diseño de los sistemas; tomando en cuenta que la tecnología basada en íconos es una de las metas actuales de CASE. En las secciones 3.5 y 3.6 se presenta un panorama de los lenguajes visuales referentes a programación de propósitos generales y los enfocados al desarrollo de sistemas. Sin entrar en mucho detalle acerca del funcionamiento de cada uno de ellos, sólo presento ejemplos de desplegados gráficos que muestran la naturaleza de ellos y sus características.

Tres clases fundamentales de Sistemas Icónicos dentro del área de procesamiento de datos son las tratadas en esta tesis: los que representan la organización y automatizan las gestiones; los que manejan una base de datos; y los que permiten desarrollar aplicaciones. El propósito de dar ejemplos gráficos es mostrar a través de su ambiente visual las diferencias entre ellos, incluyendo LIDA.

El capítulo 4 fundamenta la abstracción de datos con el modelo de datos de LIDA. La orientación del sistema al procesamiento de datos tiene que hacer uso de una base de datos, requiriendo de un modelo conceptual y otro físico. El lenguaje se basa en el modelo relacional de Codd considerando que los iconos transforman relaciones a relaciones. El modelo físico es la interfaz que me permite definir los esquemas de relación y llevarlos a una representación y organización interna. El modelo que usamos es el de Descriptor de Archivos y un módulo equivalente al Lenguaje de Definición de Datos (LDD) de tipo gráfico que permite dar de alta el diccionário y los esquemas a la base de datos. El modelo visual es similar a QBE, en donde usa tablas y el objeto visual son los datos.

El objetivo del capitulo 5 es definir la sintáxis y semántica de LIDA; mostrando cómo esta última se relaciona directamente con la abstracción procedural que se determina con la semántica de los operadores y el modelo de flujo de datos. Los iconos son clasificados en: iconos de entrada y salida; iconos que representan transacciones; iconos que representan operadores relacionales; iconos que representan funciones y procesos; y finalmente, iconos que representan el constructor condicional. Cada clase se determina por su semántica de operación o naturaleza de transformación.

El capitulo 6 de aplicaciones y resultados contiene varios casos de estudio, tanto como lenguaje de consulta como para el desarrollo de aplicaciones. El objetivo es mostrar cómo podemos resolver diversos problemas con LIDA y compararlo con otros sistemas.

El primer aspecto de aplicación de LIDA es el de lenguaje de consulta en base de datos. Se estudian algunos casos formulando las consultas con flujogramas, y comparando con QBE y SQL. Aunque QBE y LIDA son dos lenguajes de tipo gráfico, el primero muestra las relaciones directamente visibles como objetos de datos en esquema de tablas; mientras el segundo, se basa en diagramas de flujo de datos con el uso amplio de iconos. En contraste a los dos anteriores uso SQL como lenguaje de texto para dar otro punto de vista.

El segundo aspecto es tratar a LIDA como lenguaje para el desarrollo de aplicaciones, estudiando casos cuya solución se compara con el lenguaje SBA. Elegí SBA debido a su naturaleza gráfica como una extensión de QBE; en este sistema el usuario trabaja sus aplicaciones con el manejo de la información en gráficas de tablas de dos dimensiones.

Finalmente, en el capítulo 7 se dá la estructura interna del sistema, considerando tres aspectos principales: estructura interna del flujograma, método de ejecución de un flujograma e intérprete ISBL extendido. El capítulo trata de cómo es representado internamente un flujograma para su ejecución. Se analiza la interpretación del flujograma para dejar un código de cadena intermedio denominado ISBL extendido. Después de esta etapa se describe cómo el sistema entra al interprete de ISBL para resolver la cadena de código leida, encontrando de esta forma los resultados al problema planteado en el flujograma.

El trabajo concluye con un resúmen de cuales son las principales características de LIDA y comparando con algunos otros sistemas. Además se comentan algunas otras extensiones como proyectos futuros.

C A P I T U L O 1

FLUJOGRAMAS Y EL LENGUAJE ICONOGRAFICO PARA EL DESARROLLO DE APLICACIONES (LIDA).

Este capitulo tiene por objeto el dar: una reseña breve del proyecto, el concepto de flujograma y la presentación del lenguaje LIDA. Se presenta además descripción general del sistema con el procedimiento de manejo del mismo, en términos de una sesión de trabajo de LIDA.

El uso de los flujogramas dentro del tema de Programación Automática inicialmente fué propuesto por el Dr. Adolfo Guzmán Arenas como proyecto de tesis doctoral. El objetivo del proyecto fué crear un nuevo lenguaje que sirviera para plantear problemas en términos gráficos con cartas llamadas flujogramas; para qué, a partir de éstas con la metodología de PA se generaran programas que resolvieran los problemas planteados.

1.1 RESEÑA BREVE DEL PROYECTO.

El proyecto de "Programación Automática a partir de Descriptores de Flujo de Información" se inicia de forma indirecta con el trabajo [CHAP85], y consistió en el desarrollo de herramientas para un consultador y captador para una base de datos. La parte medular es el planteamiento de un esquema de organización física de datos que define la estructura de los archivos, con mecanismos de acceso sencillos y eficientes.

El modelo denominado Descriptor de Archivos es el vínculo entre entre el modelo de datos conceptual y la representación física de los datos en las estructuras de almacenamiento del sistema. Mediante el Descriptor es posible definir los esquemas de relación del modelo de datos de LIDA. En la sección 1.4 se puede ver el procedimiento manejo dentro del sistema y la descripción interna en el capítulo 4. Además el Descriptor de Archivos en el sistema proporciona el mecanismo de acceso a cada dato que se requiere para la resolución de un problema [CHAP85a].

En los sistemas para el procesamiento de datos los problemas de entrada y salida son importantes. El primero corresponde al proceso de captura de datos, que puede llegar a ser complejo debido a las diversas alternativas de interacción hombre máquina [CHAP85]. El segundo consiste en listar lotes de datos bajo una cierta clasificación y formatos para reportes. En el sistema LIDA se tiene un Captador que se maneja con la misma filosofía que el Descriptor de Archivos, basandose en ventanas y manejo del cursor (sección 1.4).

En el trabajo "Definición del Modelo de un Lenguaje de Programación (Lenguaje de Flujograma)" [CHAP86], se plantean los

primeros comentarios acerca del uso de modelos pictóricos para el desarrollo de lenguajes visuales como un nuevo y fructifero estilo de programación. Dichos lenguajes de muy alto nivel han mostrado ser muy valiosos en computación, debido a sus diversas aplicaciones: lenguajes de consulta a Base de Datos (BD), lenguajes de definición de requerimientos, lenguajes de programación, documentadores de "software" y herramienta de ayuda por computadora para ingenieria de "software".

Con respecto a la implementación del sistema, el primer antecedente de resultados parciales fué un sistema de consulta de BD basado en el lenguaje algebráico de ISBL. El lenguaje ISBL (Information System Base Language) fué creado para el sistema iteractivo de base de datos Peterlee Relational Test Vehicle desarrollado por la IBM cómo uno de los primeros sistemas de consulta. En nuestro caso, el desarrollo consistió de una colección de macros que representan operadores del álgebra relacional para ser invocados através del lenguaje algebráico definido en ISBL.

Es un hecho que en lenguajes de consulta y de desarrollo de aplicaciones en su proceso de transformación se considera diferentes niveles, que en mi opinión, podemos desglosarlas en: el lenguaje de muy alto nivel, algún lenguaje intermédio, y finalmente la codificación en un lenguaje convencional de alto nivel. Lacroix y Pirotte en [LACR80] informan cuatro tipos de lenguajes intermedios en orden ascendente de integración.

* Llamados a subrutinas, (por ejemplo: DL/1 [IBM75] y TOTAL-IQ [CINC78]).
* Extensión simple, (por ejemplo: COBOL/DML [CODA71]).
* Operadores de procedimiento, (por ejemplo: C/QUEL [STON76] y APL/EDBS [LOCH77]).
* Integración completa, (por ejemplo: PASCAL/R [SCHI77] y ADAPLEX [SMIT81]).

En esta etapa el trabajo consistió en el desarrollo de un sistema para el desarrollo de aplicaciones que tomaria al lenguaje algebráico como el medio de programación. Se consideraron extensiones para la determinación del control de flujo de constructores condicionales con evaluaciones de predicados. Así también, constructores de procesos y funciones con la especificación de formas algebráicas. Los resultados a este trabajo se consideraron para ensamblar el subsistema como el módulo de interpretación dejando a ISBL extendido como lenguaje de cadena intermedio.

La primer reseña del lenguaje de flujograma en donde se menciona como lenguaje de programación fué en [CHAP87]. En este trabajo se presentan las principales caracteristicas de flujo de de datos y modelo gráfico, considerando la abstracción de datos y la abstracción procedural como los principales elementos para definir el lenguaje de flujograma. En esta etapa se tenian definidos en el lenguaje dos tipos de líneas de flujo: línea de

flujo de datos y líneas que establecen coordinación en la ejecución de procesos. Con las líneas de flujo de datos y coordinación se forman estructuras de bifurcación, debido a que la línea de coordinación manda mensajes de un icono de evaluación de condición a dos iconos adicionales de distribución y selección.

Sin embargo, este tipo de estructura presentaba cierta dificultad en el entendimiento del programa de flujograma, no ofreciendo ventajas en el procesamiento de diferentes lotes de datos seleccionados. El problema se resolvió con el icono de bifurcación que representa el constructor condicional. Con él se muestra en los flujogramas un enrutamiento claro del programa en el sentido del flujo de lotes de datos bajo condiciones.

En el artículo "Lenguaje de Flujograma para la Consulta en Base de Datos (Un Estudio Comparativo)" [CHAP89], se muestra el lenguaje con un enfoque de lenguaje de consulta en BD. Los resultados que se tienen es la formulación de consultas con un punto de vista gráfico, dando una evaluación de los flujograma con un estudio comparativo con Query-By-Example y SQL.

1.2 FLUJOGRAMAS.

La información es uno de los productos más importantes de una oficina y su manejo incluye: creación, transformación y almacenamiento. En los sistemas de administración, la información entra a procesos de gestión sometiendose a un flujo que transita por los elementos organizacionales, y de acuerdo a las funciones de cada uno de ellos, se tienen procesos que la transforman. El aumento en la complejidad en los sistemas de información e incremento en el costo de su utilización son causas importantes para tener una automatización de las oficinas.

La automatización de la administración nos ofrece un mejor entendimiento de las funciones de las oficinas como sistema; ofreciendo a los administradores una forma más eficiente de preparar los procesos para el manejo de la información. Para tal fin, es necesario proveer al administrador de un modelo que represente la estructura de la organización, y con la especificación de los requerimientos de la información, es posible determinar el flujo de la información para predecir implicaciones en la reorganización. Además, con el empleo de éstos modelos es posible manejar la representación, la transformación y el análisis de la información

Los flujogramas en un sistema administrativo y de gestión, como herramienta gráfica sirve para representar las funciones concernientes al manejo de la información, dentro de la estructura de la organización y representar la transformación de la información. De esta forma los flujogramas los podemos utilizar para dos propósitos:

* Describir la gestión en una organización.

* Procesar información.

Cuando los flujogramas se usan para describir la estructura operacional de la organización administrativa, el usuario representa en un lenguaje gráfico en dos dimensiones, las actividades y/o las funciones que los órganos desempeñan. El flujo de la información se representa gráficamente por flechas que considerar el tránsito de la información que fluyen entre las unidades organizacionales, con la posibilidad de ser transformada en cada una de ellas. De esta manera se tiene que los flujogramas son un excelente modelo para la representación de la gestión administrativa, sirviendo para responder preguntas acerca de los procedimientos administrativos.

Cada cuadro pictórico de flujograma está constituido de símbolos gráficos que representan objetos o actividades que son esenciales en cada uno de los elementos de la organización. Los símbolos son iconos que se ligan con flechas etiquetadas con el nombre de la "forma" de los documentos, que contiene la información. En cada étapa se tiene una transición que se efectúa bajo una actividad o un conjunto de actividades, que suelen transformar y procesar la información para dar como resultado el segundo propósito de procesado de la información.

Los flujogramas como un modelo para describir el procesamiento de datos, consiste en la automatización de los procedimientos que manejan la información en alguna parte del sistema. El flujograma en esta parte expresa precisamente la relación funcional entre los documentos y el sistema. Esto quiere decir como el sistema va a efectuar las transformaciones a los documentos con un conjunto de procesos específicos. En un flujograma los procesos están descritos en una reseña gráfica que tiene iconos y líneas de flujo de datos. En los iconos se tiene la transformación de la información, y en los arcos el flujo de la misma información que sale de algunos iconos para entrar a otros.

De lo anterior se desprende la necesidad de desarrollar un lenguaje que usando el concepto de flujograma permita resolver problemas de procesamiento de datos.

1.3 EL LENGUAJE LIDA.

LIDA es un lenguaje de flujo de datos que se describe de una forma gráfica; en donde se tienen símbolos gráficos que representan operaciones sobre los datos de entrada. Los objetos de datos fluyen a través de las líneas de flujo para entrar a módulos de transformación que son los ICONOS, que tienen asociada una semántica a su imagen y con el uso de reglas sintácticas se disponen en una carta gráfica para conformar un FLUJOGRAMA.

Las principales características de LIDA son:

a) Se basa en el modelo de datos relacional.
b) Es un lenguaje de flujo de datos.
c) Su programación es visual.

En el sistema LIDA através del descriptor de archivos los esquemas que se definen son relaciones, siendo estas los objetos de datos que principalmente fluyen en las líneas de flujo de datos. El modelo de datos relacional es el enfoque de abstracción de datos de LIDA que pretende deliberadamente omitir ciertos detalles al usuario final, manteniendo sólo los que son relevantes en las aplicaciones.

El concepto de relación tiene la virtud de absorber dos clases de abstracciones: la de agregación y la de generalización, exámen que se presenta con mucho cuidado en [SMIT77]. En general, en LIDA los iconos tienen como entrada relaciones y salida relaciones. Los iconos que representan funciones aceptan relaciones y arrojan sólo un dato. Detalles del modelo de datos se dan en el capítulo 4.

El modelo de flujo de datos que tiene LIDA dá un enfoque de abstracción procedural debido a que el usuario sólo trata con iconos que representan procesos, además que el flujo de datos determina el control del flujo. Los arcos conectan a iconos que determinan la abstracción procedural como transformaciones de objetos de entrada a salidas. Las representaciones icónicas en LIDA permiten invocar a los procedimientos que encapsulan operaciones, sin tomar en cuenta la representación de los objetos, ni tampoco la implementación de las operaciones. Los iconos admiten argumentos que pueden ser: atributos, expresiones aritméticas o expresiones booleanas. Estas últimas se utilizarán como argumentos en iconos que son constructores condicionales que en su ejecución determinaran bifurcaciones en el flujo de datos mostrando una vista en paralelo (ejemplo de flujograma fig 1.1).

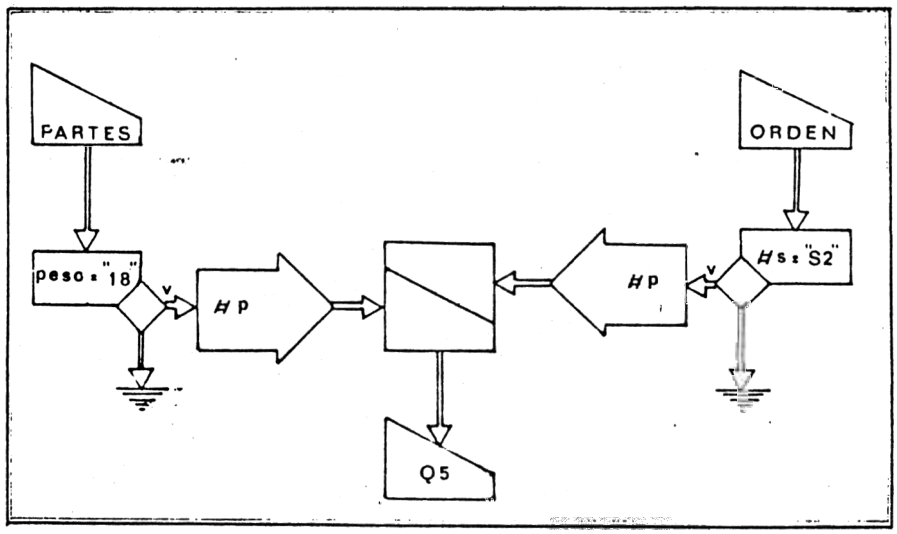


Fig. 1.1. Ejemplo de Flujorama

La figura 1.1 es un flujograma que representa el planteamiento de un problema de consulta a una base de datos. El problema es el siguiente:

Dadas las relaciones los esquemas de relación de Parte PARTE (#P,PNOMB,COLOR,PESO,CIUDAD) y Orden ORDEN (#S,#P,CANTIDAD). Queremos obtener el número de partes (#P) cuyo peso sea mayor de 18 Kgs. o bién sean suministradas por S2.

En el flujograma se puede ver que de la relación orden y parte, fluyen simultaneamente para entrar a dos iconos de selección. En estos se seleccionan las eneadas que cumplen con peso igual a 18 y proveedor igual a S2. Posteriormente, cómo la operación de en medio es una unión y requiere que los esquemas de relación sea iguales, sólo proyectamos el atributo #P con el icono flecha, para que el icono unión procese las dos relaciones unarias y arroje el resultado en la relación Q5.

Un rasgo distintivo de LIDA es su programación visual, que se establece en la edición y puede persistir hasta la documentación. El usuario sólo trata con la disposición de iconos en la pantalla y conecciones de líneas. Considerando además cuando la semántica del icono lo requiere añadir texto que son argumentos; para ver en detalle el procedimiento de edición ver la siguiente sección en el modo LIDA. Dadas estas características de edición gráfica puedo decir que el sistema LIDA puede ser visto cómo un sistema de Diseño Asistido por Computadora para la programación.

En LIDA la sintáxis esta dada por las líneas de flujo que conectan a los iconos. Según un resúmen de iconos que puede ser consultado en el apendice A son:

* Iconos terminales de los cuales sólo entran líneas de flujo de datos e iniciales de las cuales salen. Estos son escribe relación y lee relación.
* Iconos que representan operadores binarios de los cuales siempre estan esperando dos relaciones de entrada y arrojan una de salida.

* Iconos unarios que aceptan una relación y dan como resultado otra relación.

* Iconos que aceptan una relación y bajo una condición establecida con una expresión booleana dividen las trayectorias de flujo en dos una verdadera y otra falsa.

* Iconos funciones que aceptan en su entrada una relación y arrojan un valor que se manda a un icono rectángulo. Esta será identificada por un nombre.

Observación: El comentario de que sale una sola línea de cada icono es sólo por facilidad de entender la sintáxis. En realidad de los iconos pueden salir varias líneas de flujo,para apuntar a varios procesos; pero sólo portando la misma relación.

1.4 DESCRIPCION GENERAL DEL SISTEMA LIDA.

Con el fin de tener un lenguaje con las caracteristicas mencionadas en la sección 1.3, fué necesario integrar un sistema que tuviera la capacidad de definir los esquemas de relación que estructure una base de datos y la construya con un captador de datos. Además, por medio de un editor gráfico tuviera la posibilidad de generar flujogramas que puedan ser traducidos a un código intermedio de cadena; para ser posteriormente interpretados por el intérprete del lenguaje ISBL extendido. Por tal razón el sistema LIDA contiene fundamentalmente los siguiente:

* Un módulo denominado Descriptor que permite capturar la definición de los datos y estructurar los esquemas de relación. El procedimiento es por despliegue de ventanas, manejo del cursor y actualización de datos. Este proceso es equivalente al llevado a cabo por un Lenguaje de Definición de Datos LDD.

* El módulo anterior esta completamente relacionado con un esquema general que vincula el modelo conceptual y el modelo físico de datos. La estructura se denomina Descriptor de Archivos y proporciona también, el mecanismo de acceso a la base de datos.

* Un módulo capturador de datos que permite construir la base de datos. El enfoque también es de ventanas y el formato de columnas lo proporciona la definición de los datos dados por el módulo descriptor.

* Un editor gráfico para los flujogramas. El editor presenta una ventana principal para los diagramas y ventanas secundarias de comandos y menúes de iconos.

* Un intérprete de código de cadena. Que es el lenguaje intermedio del sistema denominado ISBL extendido.

1.4.1 Aspectos Generales del Sistema.

El sistema LIDA puede instalarse en una microcomputadora compatible IBM, con capacidad de 512Kb de memoria mínima, monitor del tipo VGA, EGA, CGA o Hércules; disco duro de 10MB en adelante (aunque no es necesario por motivos de capacidad y rapidez recomendamos disco duro), una unidad de disco flexible 5 1/4" es necesaria. LIDA corre bajo el sistema operativo DOS versión 2..0 en adelante.

El sistema consta de archivos del sistema, de utileria y archivos generados:

Archivos del Sistema

LIDA.EXE Sistema LIDA ejecutable.

Archivos de Utileria

LIDA.CFG	Configuración del sistema.
DESCRIP.DES	Archivo de Descriptores.

Dentro de LIDA.CFG se almacena la configuración del sistema, que incluye eltipo de monitor y la ruta del directorio del archivo de descriptores. DESCRIP.DES contiene los descriptores de archivos.

Archivos Generados.

*.PRN	Impresiones formato ASCCI.
*.LID	Flujogramas.
*.LIP	Procedimientos en lenguaje intermedio
* !	Archivos temporales que se generan cuando se ejecuta un flujograma.
	Base de Datos.

4.2 Manejo del Sistema.

Si se encuentra en disco duro y subdirectorio correspondiente entonces escriba delante del prompt:

C>LIDA y presione <ENTER>.

Espère a que el sistema operativo carge LIDA en la memoria y aparezca la pantalla principal que consiste en la ventana del sistema (Fig. 1.2). La pantalla principal se divide en cuatro partes: una línea en la parte superior de la pantalla que esta dedicada al menú principal, el espacio para las ventanas de trabajo y captura de datos, una línea para el despliegue de mensajes; y una última línea que muestra el estado relacionado el teclado con <CAPS-LOCK> o <NUM-LOCK>.

1. Movimiento del Cursor.- En la mayor parte del sistema, el cursor es representado por una barra perfectamente distinguible, con la libertad de movimiento vertical u horizontalmente sobre la pantalla y movida con teclas <ARRIBA>,<ABAJO>,<IZQUIERDA> y <DERECHA>.

2. Actualización de Datos.- Dentro del procedimiento de captura de datos, las teclas <HOME> y <END> ponen el cursor en la primera posición de la línea donde estamos capturando el dato o al final respectivamente. Las teclas <DEL> e <INS> ponen al sistema en modo de borrado e inserción, colocando para este último en la línea de estado "INS". Con las teclas <Pg Up> y <Pg Dn> es posible posicionar el cursor al principio o final de la ventana.

3. Alta de Registros.- Cuando se requiere realizar una alta de cualquier registro que contenga información, sólo debe oprimirse la tecla <INS>.

4. Baja de Registros.- Para dar debaja cualquier registro, debe oprimirse la tecla <DEL>, previamente habiendo posicionando el cursor sobre el registro a eliminar, el sistema generalmente enviará un mensaje para confirmar la baja del registro, si usted oprime la letra <S> el registro es eliminado, de lo contrario usted sólo debe omprimir la letra <N>.

5. Salida.- Para salir de cualquier opción, es necesario oprimir <ESC> en cualquier momento si desea abandonar el sistema. El sistema enviará un mensaje para confirmar la salida.

La línea dedicada al menú principal muestra cinco opciones:

Descriptor Captador Intérprete LIDA Configuración

Alguna de las opciones puede ser elegida por medio del movimiento del cursor, considerando el modo de operación del sistema.

DESCRIPTOR

Cuando posicionamos el cursor en Descriptor y damos <ENTER> el sistema abre un submenú que muestra dos opciones:

a) Mantenimiento
b) Reportes

En la figura 1.2 damos la pantalla correspondiente a la ventana del descriptor. Con sólo posicionarse en alguna de las dos opciones y dar <ENTER>, el sistema abrirá una nueva ventana. La primera que corresponde al la Fig. 1.4, es la ventana de mantenimiento de descriptores. Esta ventana despliega los nombres de las relaciones que están dadas de alta, con la posibilidad de una actualización. La Fig. 1.5 Es la ventana de reporte de descriptores que se detalla en el inciso b.

a) Mantenimiento

Oprima <ENTER> sobre la primera opción y aparecerá una ventana que muestra los nombres de los archivos contenidos en el archivo de descriptores.

* Alta, Baja y Modificación de los Descriptores de Archivos.

Para generar un nuevo descriptor, sólo teclee <INS> sobre la ventana de los archivos del descriptor; requiriendo a continuación el nombre del archivo (nombre de la relación), captúrelo y la inserción se realizará automáticamente (ver Fig. 1.4). Para definir el esquema del archivo sólo coloque el cursor sobre el nombre del descriptor que acaba de dar de alta y de <ENTER>, en ése momento el sistema abrirá una nueva ventana mostrando el espacio para definir la estructura del archivo o bién el esquema de relación (Fig. 1.6). La ventana esquema de relación incluye los siguientes campos:

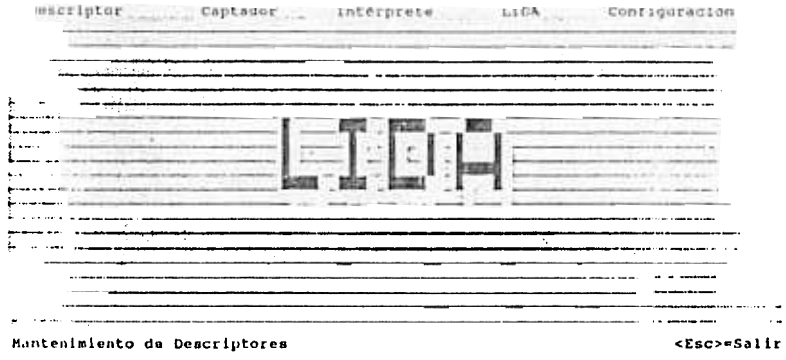


Fig. 1.2 Ventana del Sistema

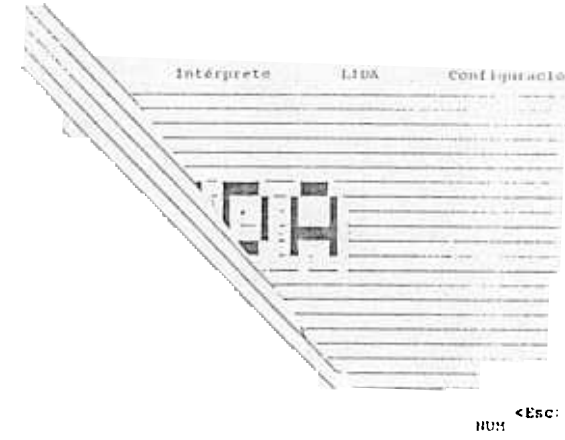


Fig. 1.3 Ventana del Descriptor

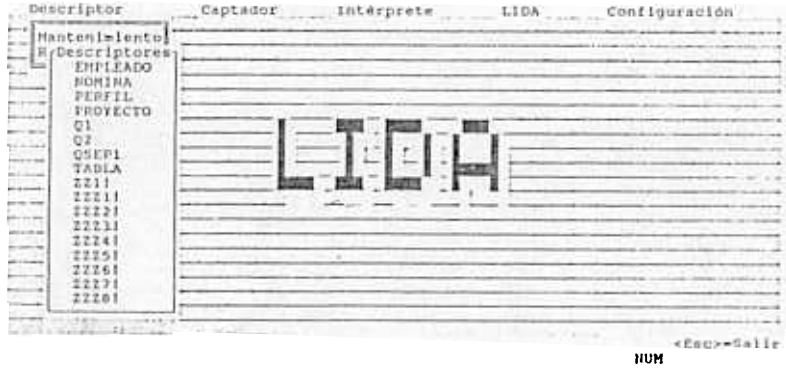


Fig. 1.4 Ventana de Mantenimiento de Descriptor

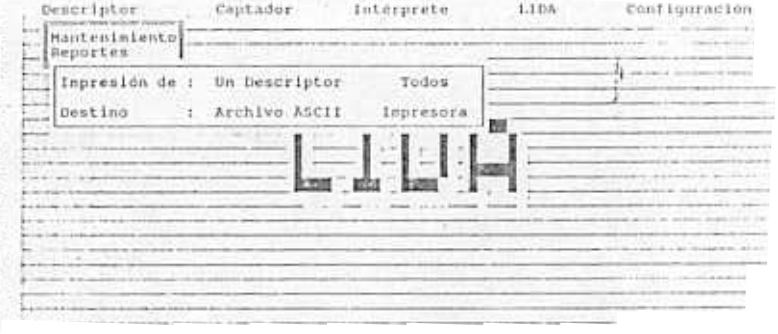


Fig. 1.5 Ventana de Reporte de Descriptores

No.	Número consecutivo del campo.
Campo	Nombre del Campo (Atributo de la relación)
Tipo	Tipo (A = Alfanumérico, N = Numérico)
Lon.	Longitud del campo.

En este momento usted puede comenzar a capturar los atributos tecleando <INS> o <ENTER>, el sistema abre el espacio para la captura del nombre del campo terminando siempre con un <ENTER>.

El atributo quedó insertado y como puede observar los campos de tipo y longitud tienen valores iniciales. Para cambiarlos posicione el cursor sobre la columna respectiva y teclee <ENTER>. El tipo sólo puede ser "A" o "N" , y la longitud entre el rango de 1 a 78. Para terminar teclee <ESC> y responda <S> a la pregunta que hace el sistema.

Si desea eliminar un archivo, conjuntamente con su descriptor, posicione el cursor sobre el archivo a eliminar y teclee <DEL>, el sistema hará una pregunta para confirmar, debiendo contestar <S> o <N>. Para modificar el contenido de cualquiera de los descriptores, posicione el cursor sobre el nombre del archivo, dando enseguida <ENTER> y de la misma forma que usted pudo cambiar los valores iniciales proceda de la misma forma. Es posible añadir más campos e insertar entre dos consecutivos.

Si al finalizar la descripción del archivo, desea realizar la captura de los datos oprima <F2>, y en ese momento se abrirá la ventana de captura, pasando al modo Captador.

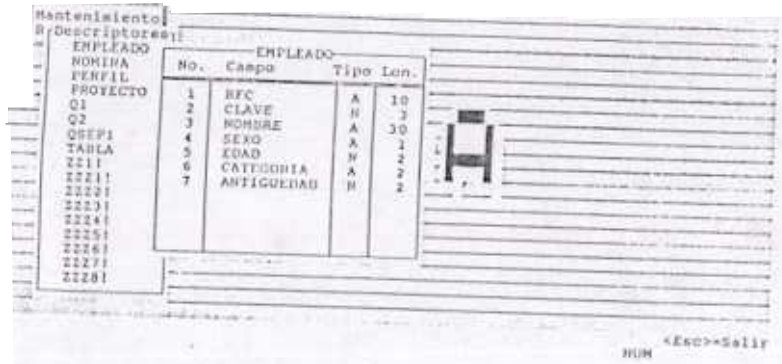
b) Reportes

Esta opción abre la ventana de reportes de descriptore Fig. 1.5 con dos menúes horizontales. Los reportes se realizan sobre la información almacenada por el archivo de descriptores pudiendo seleccionar el reporte de todos los descriptores o sólo uno. Cuando se opta por la impresión de un solo descriptor, entonces se abrirá otra ventana mostrando todos los descriptores que existen, cómo puede verse en la fig. 1.7 Ventana de Reporte de un Descriptor. En seguida, para que el sistema imprima el esquema de relación, sólo seleccione el descriptor moviendo el cursor y teclee <ENTER>.

El segundo submenú determina el destino de la impresión con la posibilidad hacia impresora o un archivo de formato ASCII. El sistema pregunta si la impresora esta lista para el primer caso o si el usario pretende sobreescribir en un archivo, considerando que el usuario sólo debe dar el nombre y el sistema pone la extensión PRN.

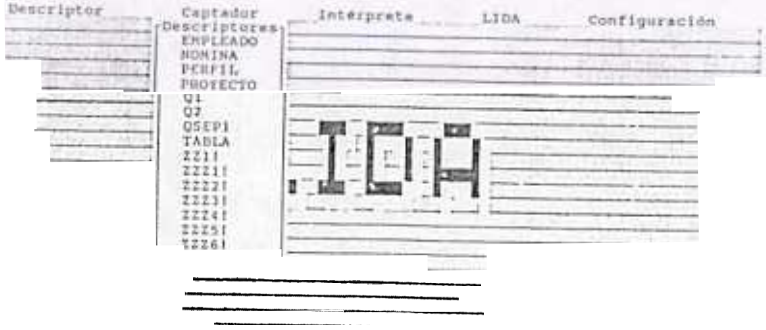
CAPTADOR

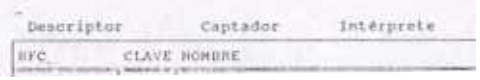
El modo Captador es un sistema de captura de datos que nos permite realizar altas, bajas y cambios a los registros de las bases de datos. Una vez que ya hemos definido el esquema de

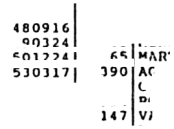


ig. 1.6 Ventana del Esquema de Relación

Fig. 1.7 Ventana de Reporte







relación através del descriptor de archivos, es posible capturar datos en un esqueleto de forma que se genera automáticamente por la definición del descriptor. El primer paso es colocar el cursor en la opción de captador para que el sistema abra una ventana de las relaciones que estan definidas (Fig. 1.8).

Es conveniente recordar que en el descriptor definimos los atributos que conforman el esquema de relación. A partir de éste el captador genera una ventana de captura de datos constituida por varias columnas, cada una de ellas tiene en el encabezado el nombre del atributo que se definio en el descriptor.

Elegida la relación que se quiere construir o actualizar, con un <ENTER> pasamos al modo de captura desplegando el sistema una ventana de captura de datos cómo puede verse en la fig. 1.9. En el modo de captura el cursor se puede mover con las flechas libremente através de la ventana, y con <PgUp> y <PgDn> podemos ir al principio del archivo o al fin respectivamente. Si usted posiciona el cursor sobre alguna columna y teclea <INS>, el sistema abrirá el espacio para aceptar datos del nuevo registro.

El primer dato que se requiere capturar es aquel en el que se encuentra el cursor y en el momento de oprimir <INS> los demás campos deberán capturarse posteriormente. Para continuar posicione el cursor en cada una de las columnas adyacentes y comience a teclear el dato de inmediato. Puede utilizar las teclas <BS> para borrar caracteres, o <INS> para insertarlos, con las flechas <DER> e <IZQ> el cursor se puede mover en ambas direcciones.

Para dar debaja algún registro, posicione el cursor sobre dicho registro y teclee <DEL>, el sistema mandará un mensaje para confirmar si el registro señalado por el cursor se desea eliminar, responda afirmativamente y la baja se efectuará en seguida.

INTERPRETE

Cuando usted se posiciona en Intérprete y teclea <ENTER> inmediatamente el sistema abre una ventana con el envio del prompt ISBL>. Este indica que el sistema esta listo para aceptar cadenas del lenguaje ISBL extendido. Cómo puede observar de la tabla 1.1 el intérprete maneja comandos que sirven de ayudas a la operación del sistema.

El editor del intérprete, cuenta con las facilidades de las teclas para realizar la captura de los comandos. Cuando usted edita una cadena y teclea <ENTER> el sistema la interpreta y ejecuta (para detalles en el proceso interno ver capítulo 7). Cualquier error de sintáxis detectado por el intérprete es avisado al usuario por medio de un mensaje sobre la pantalla y una flecha indicando el caracter que originó el error, si en ese momento se requier corregir la instrucción, sólo pulse <ARRIBA> y la línea volverá a aparecer para que usted proceda a su correccción.

TABLA 1.1 Lista de Operaciones y Comandos del Intérprete.

OPERACION	SIMBOLO	EJEMPLO
Asignación o Copia	=	Empleado=Profesor
Unión	+	Empleado=ProfesorA+ProfesorB
Intersección	.	Empleado=ProfesorA.ProfesorB
Diferencia	-	Empleado=Profesor-Investigador
Junta	*	Empleado=Profesor*Investigador
Selección	:	Empleado:(Sueldo>300)
(Parte Falsa)	,	Viejos =Empleado:(Edad>60), Jovenes
Proyección	%	Empleado%Nombre,Sueldo,RFC
Renombra Atributo	->	Empleado%Nombre->NomEmp
Proceso	[ ]	Empleado[Sueldo=Sueldo*3.1416]
Ordena (Ascendente)	#	Empleado#Sueldo,A
(Descendente)		Empleado#Sueldo,D
Permuta Llave	~	Empleado~Llave,5
Máximo	MAX()	@Z =MAX(Empleado,Sueldo)
Mínimo	MIN()	@Z =MIN(Empleado,Sueldo)
Promedio	PRO()	@Z =PROM(Empleado,Sueldo)
Suma	SUM()	@Z =SUM (Empleado,Sueldo)
Número de Registros	REGS()	@Z =REGS(EMpleado)

COMANDOS:

$LISTA	Genera un reporte hacia algún dispositivo (Consola, Impresora o Archivo ASCII).	
	$LISTA Empleados	Envía relación Empleados hacia la consola.
	$LISTA Empleados < CON	
	$LISTA Empleados < LPT1	
	$LISTA Empleados < TEMP	Envía relación Empleados Archivo TEMP.PRN.
$DESC	Abre menú de mantenimiento de descriptores.	
$CLS	Limpia la pantalla de edición.	
$DESPV	Despliega las variables de memoria.	
$FDECS	Define número de decimales a usar Ejemplo: $FDESC 2.	
$EJEC	Ejecuta un programa .LIP Ejemplo: $EJEC Nomina.	
FIN	Sale del intérprete.	

Manejo de variables de memoria:

@X=30.78	Asigna 30.78 a la variable de memoria X.
@X='HOLA'	Asigna la cadena 'HOLA' a la variable X.
?X	Editar el contenido de la variable X.
@A=&B	Asigna el contenido de B en A.

Observaciones:

* La Selección utiliza expresiones con los siguientes operadores:

(O "O" lógico), (Y "Y" lógico), < menor que), (> mayor que), (= igual), (<= menor o igual), (>= mayor o igual), (<> diferente), (() asociación).

Ejemplo:

```
EMPLEADO: (sueldo <= 3 000. ) Y  (comp > 1 000)
          ( (edad > 50 ) Y (sexo = 'M') )
```

En la selección es posible usar expresiones aritméticas dentro de la expresión booleana. Siempre deben encerrarse entre paréntesis.

Ejemplo:

```
IMPUESF = EMPLEADO:(sexo='F' Y ((sueldo+presta)*.78) > 30000)
```

* En la operación proceso y asignación de variables, es posible utilizar expresiones aritmeticas con los siguientes operadores:

(+ suma), (- resta), * multiplicación), (/ división), (() asociación).

Ejemplo:

```
EMPLEADO=EMPLEADO[sueldot=sueldo*25. + presta/1.25 * bonos]

                .7566)/23.178 * (1.23 + 3.12)
```

* Una instrucción ISBL no puede tener una longitud mayor a los 72 caracteres (que es la longitud de la ventana del intérprete dispuesta para la captura de instrucciones), pero sin embargo es posible expander una instrucción hasta los 256 caracteres), haciendo uso de las variables de memoria.

Ejemplo:

```
@compensacion= '34.56 + presta/6* (interes+2.5)'

EMPLEADO=EMPLEADO[sueldot=sueldo*&compensacion].
```

LIDA

La opción de LIDA abre una pantalla que le permitirá editar gráficamente un flujograma. La pantalla esta dividida en una ventana superior destinada para la representación de los flujogramas y una ventana inferior para: los comandos del editor de flujogramas, el menú de los iconos de los flujogramas, captura de datos y envío de los mensajes de error.

En la primera ventana se localiza un cursor en forma de cruz, que indica en donde va a colocarse el icono correspondiente. En la segunda ventana se encuentran en primer lugar los comandos que permiten editar un flujograma (ver fig. 1.10 Ventana de Flujograma).

* Manejo del Editor de Flujogramas.

La edición de un flujograma en LIDA se lleva a cabo en la ventana superior con el posicionamiento del cursor_cruz que puede moverse con las flechas del teclado:

<IZQ>	Mueve el cursor hacia la izquierda.
<DER>	Mueve el cursor hacia la derecha.
<ABAJO>	Mueve el cursor hacia abajo.
<ARRIBA>	Mueve el cursor hacia arriba.
<CTRL> <IZQ>	Mueve el cursor hacia la pantalla izquierda.
<CTRL> <DER>	Mueve el cursor hacia la pantalla derecha.
<PGDN>	Mueve el cursor hacia la pantalla superior.
<PGUP>	Mueve el cursor hacia la pantalla inferior.

Con el movimiento del cursor y los comandos de edición es posible comenzar a armar un flujograma.

* Comandos del Editor de Flujogramas.

Para tener acceso a cualquier comando que se encuentra en la ventana inferior de la pantalla, basta con teclear la letra mayúscula que aparece en el comando que se quiere elegir. Por ejemplo ver en la fig. 1.11, cuando se eligío el comando Icono <I>, aparece en la ventana inferior un menú de iconos que pueden ser escogidos para colocarse en el flujograma. Para abandonar la opción LIDA solo es necesario teclear <ESC>.

Icono.- Este comando permite acceder al menú de iconos disponibles en el lenguaje LIDA. Con la ayuda de la flechas <IZQ> y <DER> es posible seleccionar alguno de los iconos, para que poco después se coloque en el lugar del cursor-cruz con <ENTER>. Para tener acceso al resto de los iconos es necesario teclear <+> y de forma similar serán mostrados los demás.

teXto.- Como se sabe , los iconos de LIDA incluyen ciertos parámetros y argumentos, que representan nombres de relaciones, expresiones aritméticas o expresiones booleanas. Por medio de este comando es posible asignar los argumentos a cada uno de los iconos que lo requieren. Una vez que se accede al comando el sistema muestra dos opciones: teclear <N> para asignación de nombre de: relació, proceso y función; teclear <P> para capturar argumento que incluyen expresiones aritméticas y booleanas. Para concluir teclee <ENTER>.

traza Linea.- Para poder dibujar las líneas de flujo de datos entre los iconos, es necesario posicionar el cursor_cruz sobre el icono de donde partirá la linea, para posteriormente teclear <L>, y finalmente posicionar el cursor_cruz en el icono destino, dibujandose la línea inmediatamente después de dar <ENTER>.

Tipo línea.- El comando descrito anteriormente, esta directamente relacionado con éste. Cada vez que se dibuje una línea de flujo de datos será trazada con el tipo actual de línea. Existen sólo dos tipos de líneas: continua y punteada, que se pueden seleccionar con sólo teclear <T>.

Borra.- Para eliminar algún elemento del flujograma, e incluso el flujograma mismo, debe accederse a este comando. Borra incluye tres opciones: borra icono, borra línea y borra flujograma. De la misma forma como accedimos a los comandos Icono y traza Linea en el caso de borrado se borra icono o línea respectivamente. Si es el flujograma el que se quiere eliminar, entonces basta con teclear <F> para que la ventana quede en limpio.

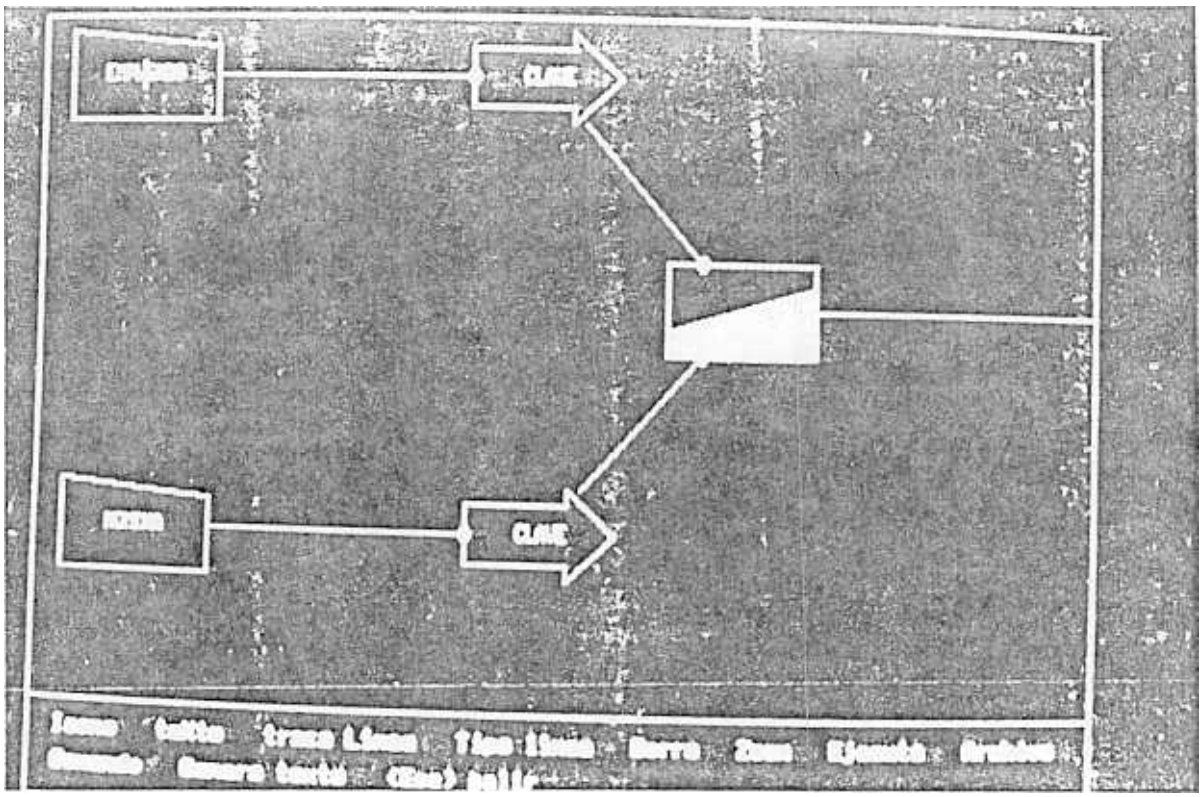
Zoom.- El comando Zoom permite visualizar el puerto completo dentro de la ventana superior y viceverza puede visualizar una parte del puerto.

Ejecuta.- En el momento en que se ha terminado de editar el flujograma, es posible ejecutarlo. Para ello sólo se requiere acceder a este comando.

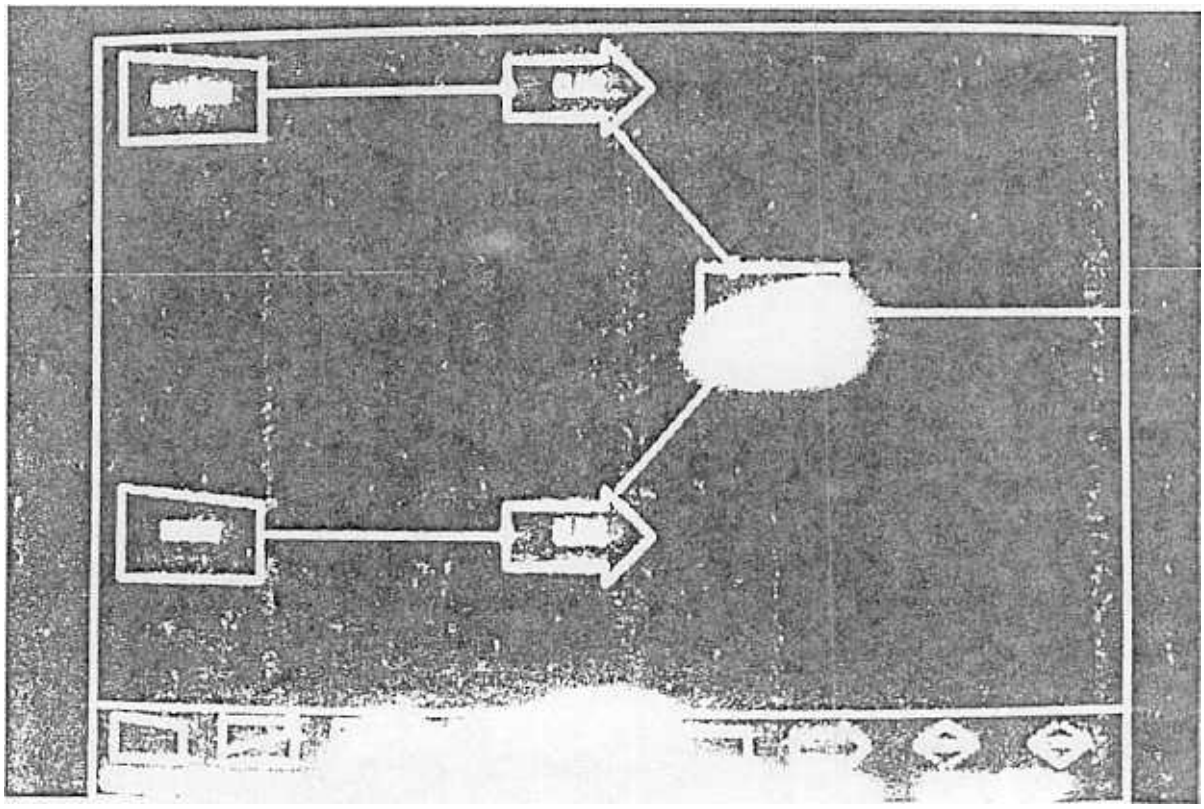
Archivo.- Este comando permite desplegar el contenido de un directorio, cargar flujogramas al sistema o bién grabarlos en disco, para lo cual cuenta con las siguientes opciones: Directorio, Carga archivo y Salva archivo. El acceder a cada una de las opciones es atravès de las letras mayusculas <D>, <C> y <S> respectivamente.

Comando.- Todos los comandos e instrucciones que acepta el intérprete del lenguaje de cadena ISBL extendido, también pueden ejecutarse desde LIDA. Para ello accese la opción y proceda a capturar la cadena ISBL extendido, de la misma forma como se realiza en el intérprete.

Genera código.- Existe una forma de escribir el programa que genera el flujograma en lenguaje ISBL extendido, y tomar éste como un procedimiento ejecutable, vía comando $EJEC o vía icono procedimiento. Al acceder éste comando el sistema requerirá del nombre del archivo procedimiento, su captura puede realizarse de la misma forma cómo en la opción Salva archivo. Una vez que se capture el nombre del archivo, es necesario ejecutar el flujograma para que el código sea generado.



Ventana de ujograma con Menú



Ventana de Flujograma con Menú de conos

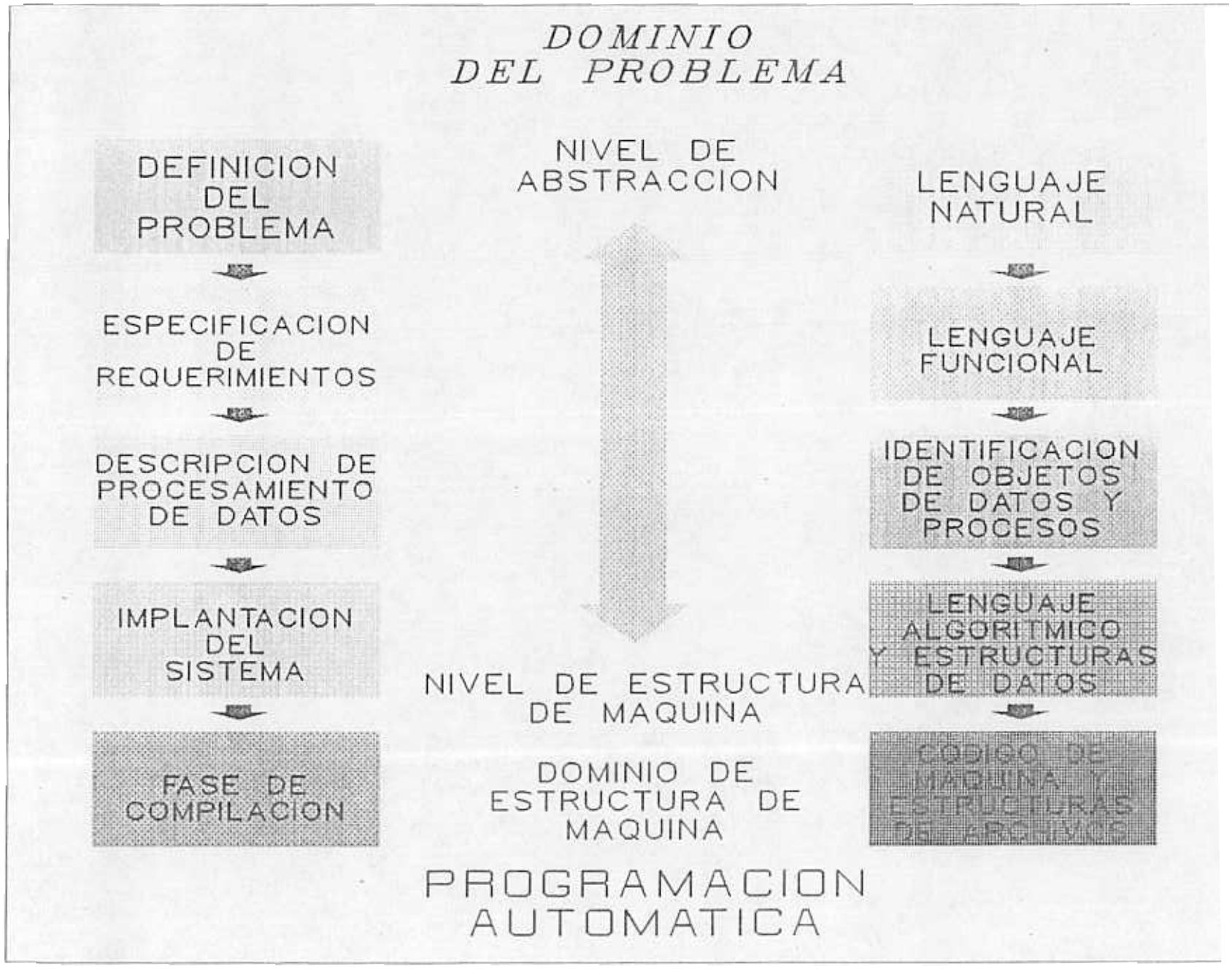
C A P I T U L O 2

PROGRAMACION AUTOMATICA

Como se mencionó en el capitulo anterior, el trabajo se ubica en la Programación Automática (PA); razón por la cual, en este capítulo presentamos su enfoque y la naturaleza del tema. La PA es una rama de la Inteligencia Artificial, cuya metodología para la síntesis de programas, es usada para la creación de Lenguajes de Muy Alto Nivel (LMAN). Los LMAN son caracterisados por dos aspectos principales: el de abstracción procedural y el de abstracción de datos. Con respecto al primero tenemos el control de flujo y la abstracción en los procedimientos; mientras que, para el segundo se tiene la definición de una estructura y el comportamiento de los datos. En este capítulo, con el objetivo de dar un marco teórico al lenguaje LIDA, doy una clasificación de los LMAN con énfasis en los lenguajes para la automatización de las oficinas.

El concepto de Programación Automática ha sido un poco difuso por lo difícil de definir el enfoque y el alcance del tema. La especificación inicial de la PA como programa escrito en un lenguaje de alto nivel -que se mantuvo hasta inicios de los 70's, mientras permaneció referida al concepto de compilador [SAMM72]- se ha transformado en un concepto con un alcance mas ambicioso. Donald Knuth establece en [KNUT74], que la idea principal de la PA es: a partir de la definición de un problema por parte del usuario, se desarrollan sistemas que construyan automáticamente programas para la solución de los problemas planteados. De esta forma un sistema de PA es aquel que lleva a cabo parte de la tarea de programación que el programador usualmente efectúa en la solución de un problema.

Hasta la fecha, la principal contribución de la PA ha consistido en el desarrollo de sistemas con orientaciones particulares o aplicaciones específicas. En cada una de ellas, es necesario establecer diferentes clases de requerimientos propiciando el uso de diversas técnicas de solución. Sin embargo, uno de los principales objetivos de la PA, es la creación de sistemas que manejen los requerimientos en términos generales y adecuar las necesidades del usuario en la generación de un programa único que resuelva cada problema particular. Por lo tanto es conveniente que los sistemas proporcionen lenguajes de interacción hombre-máquina que permitan niveles de discurso de abstracción más alta, siendo necesario que en la realización de dichos sistemas se construyan varias capas de "software" que realicen transformaciones a lenguajes intermedios generando "software" nuevo.


DOMINIO
DEL PROBLEMA
NIVEL DE
ABSTRACCION
DEFINICION
DEL
PROBLEMA
ESPECIFICACION
DE
REQUERIMIENTOS
DESCRIPCION DE
PROCESAMIENTO
DE DATOS
IMPLANTACION
DEL
SISTEMA
FASE DE
COMPILACION
LENGUAJE
NATURAL
LENGUAJE
FUNCIONAL
IDENTIFICACION
DE OBJETOS
DE DATOS Y
PROCESOS
LENGUAJE
ALGORITMICO
Y ESTRUCTURAS
DE DATOS
CODIGO DE
MAQUINA Y
ESTRUCTURAS
DE ARCHIVOS
NIVEL DE ESTRUCTURA
DE MAQUINA
DOMINIO DE
ESTRUCTURA DE
MAQUINA
PROGRAMACION
AUTOMATICA

El desarrollo de un sistema de "software" siempre se inicia con la especificación de lo que se quiere hacer, para proseguir con una serie de etapas en donde se dan descripciones más detalladas que se relacionan con la concepción de código, como puede observarse en la figura 2.1.

En la primera etapa, se atiende fundamentalmente al planteamiento del problema expresando el sentido y el objetivo del sistema en lenguaje natural. A partir de este, con un lenguaje funcional se facilitan las especificaciones de los requerimientos describiendo los procesos fundamentales del sistema de "software" en términos de sus entradas y salidas; considerando a los lenguajes gráficos excelentes para el modelado funcional de los procesos, como puede observarse en la técnica de Gane y Sarson (Fig. 3.5). Establecida esta etapa es necesario pasar a una tercera, la cual consiste en la conversión a descripciones orientadas al procesamiento de datos, que identifica y diseña los objetos de datos, los procedimientos y sus interrelaciones. En su turno, la cuarta etapa, ofrece una implementación del sistema del diseño abstracto a una realización mas concreta, que lleva el problema a un lenguaje algorítmico con sus estructuras de datos lógicas y almacenamiento físico. Para dar finalmente una solución de compilación del programa que produce código objeto ejecutable y las estructuras de archivos asociadas.

Automatizar el proceso de desarrollo de "software" significa que la computadora participe en la tarea de elaboración del mismo, transfiriendo a la máquina el proceso de transformación de alguna o todas las etapas mencionadas. Lo anterior permite que el programador se aparte del detalle de "Como se Debe de Realizar el Trabajo", enfocándose únicamente a "Que se Quiere Hacer". En esta dirección históricamente la evolución de la automatización de "software" se inició con la transición de los lenguajes de máquina a los lenguajes de ensamblador, sucediendo después, los lenguajes de programación de alto nivel. En la actualidad siguiendo la espiral de desarrollo, el progreso de la tecnología de "software" plantéan sistemas mas complejos y variados como son: los sistemas para diseño automático en ingeniería de "software", los sistemas basados en conocimiento, los lenguajes de muy alto nivel y los sistemas manejadores de base de datos, entre otros. Dando como resultado una corriente tecnológica que se destaca por cambiar la forma de "Cómo Hacerlo" a especificaciones de "Qué se Quiere Hacer".

Con la PA el programador ha sido relevado de la tarea de determinar los detalles que se relacionan directamente con la construcción de código, permitiendo incrementar la eficiencia en el desarrollo de los sistemas. Es un hecho que cuando un programador o usuario final dirige su atención a temas más relevantes a su problema, es posible llegar a soluciones mas rápidas y correctas. Liberar al programador de los detalles de la implementación, permite trabajar en un nivel conceptual más alto, pudiendo escribir y leer los programas mas sencillamente en el contexto del dominio del problema. Por consiguiente se tiene que la PA ha brindado beneficios bastantes claros como son:

a) Reducción de errores en la operación de los sistemas

b) Incremento en la complejidad de los sistemas que se implementan

c) Un uso más eficiente de los recursos que se mantienen en el ciclo de vida de los sistemas.

Los beneficios que brinda la PA conducen a un rendimiento mayor de los sistemas, aumentando la confiabilidad del "software" y disminuyendo el consumo de tiempo extra que se lleva la elaboración detallada del código. Por otro lado, la disparidad que se tiene entre la conceptualización en el dominio de planteamiento de los problemas y los medios de expresión que tiene el programador, ha llegado a ser otro problema que evita que los sistemas funcionen en la línea proyectada por los requerimientos definidos o por el compromiso original de las especificaciones.

En conclusión, el objetivo de brindar al programador o usuario la facilidad de una comunicación con la máquina en una denotación mayor de lo "Qué Quiere" reduciendo la forma de "Cómo Hacerlo", conduce a que la máquina realice una gran parte del trabajo de programación dejando al usuario solo el planteamiento de los problemas, se satisface con la Programación Automática.

2.1 PROGRAMACION AUTOMATICA E INTELIGENCIA ARTIFICIAL.

La programación de las computadoras ha sido considerada históricamente como una actividad reservada para el humano. Se ha asumido que los programadores deban ser los autores de líneas secuenciales de código para que la máquina las ensamble y lleve a cabo los resultados que se desean. Desde la decada de los setentas algunos investigadores [McCu77-79], [BLAZ77-78], [PRYE79], [SHU85] han estado explorando la posibilidad de que el código pueda ser escrito automáticamente. Con la PA se pretende que los programas puedan ser construidos y ejecutados automáticamente por la máquina, los usuarios solo deben de establecer ciertas metas que pueden ser dadas de alguna de las formas siguientes:

a) Especificación Formal del Comportamiento de la Entrada y de la Salida.

b) Descripción Global e Informal de las Especificaciones por parte del Usuario. Puede considerarse una interacción de dialogo con la máquina, desde el punto de vista de Computadora que Asiste al Diseño (CAD).

c Inclusión de Ejemplos del Comportamiento deseado o de otra Información Fragmentada.

d) Inclusión de Operadores de Alto Orden como parte de algún lenguaje de abstracción alta.

La finalidad de un sistema de PA es aceptar la especificación de requerimientos por alguna de sus formas he implica que lleve a cabo algunas tareas para la construcción de código ejecutable:

1 El establecimiento de las estructuras de datos.

2 La construcción de secuencias de comandos con ciclos y ramificaciones.

3 La construcción de subrutinas apropiadas.

4 Completar algunas otras tareas que se relacionan con la construcción total del programa.

Con lo anterior el humano obtiene un trabajo útil de las máquinas, sin la necesidad de cuidar de la elaboración del código del programa en el sentido tradicional. El usuario interactúa con la máquina en forma de diálogo en un nivel de abstracción alto, para dejar que efectúe el trabajo requerido, ejecutando actividades que algunas veces no son estructuradas y entendidas completamente; reafirmando así, la ubicación de la PA dentro del campo de la Inteligencia Artificial. La PA usa técnicas para resolver problemas referentes a especificaciones ambiguas, planeación en el desarrollo del "software", organización de los programas y representaciones eficientes de las estructuras de datos abstractas. Usando áreas de aplicación como son: procesamiento del lenguaje natural, sistemas basados en conocimiento, planeación y solución de problemas.

La investigación en PA se realiza de acuerdo con dos metas fuertemente interrelacionadas. La primera, atañe a probar que un programa deba de trabajar como fué pensado y creado; correspondiendo a lo que se denomina como verificación de programas. Mientras la segunda, generación de programas, toca a la síntesis de programas que probablemente deben actuar en una forma determinada.

La mayoría del trabajo de verificación de programas se basa en la técnica propuesta por Floyd, que consiste en incluir aseveraciones en una carta del flujo del programa para ser verificados. Originalmente las aseveraciones que se proponen en el programa han sido proporcionadas por el humano. Sin embargo, la tendencia dentro de la investigación de Prueba Automática de Teoremas considera la generación automática de las aseveraciones para ser probadas por el humano o la máquina indistintamente. Sin embargo, el hombre las debe seguir proporcionando; si las proporciona la máquina debe de derivarlas de algún conjunto más basico de aseveraciones hechas por el humano.

La generación de programas constituye un trabajo en donde la construcción del plan requiere de razonamiento, sentido común y deducción. El caso de la generación de programas es un tronco del

cual la Programación Automática se lleva por dos vertientes: los Lenguajes de Muy Alto Nivel (LMAN) y los Sistemas Basados en Conocimientos (SBC) [BLAZ73].

En principio, se considera que una metodología difiere de la otra por las expectativas que ofrecen cada una de ellas en el momento de entrar en el proceso de desarrollo de "software". La naturaleza misma de cada sistema de PA y la complejidad de la tecnología que el mismo empleará, serán dos consideraciones posteriores que diferencian a las dos aproximaciones. La construcción automática de "software" con los LMAN ha mantenido una línea de evolución más sistemática. Mientras que, con los SBC el empleo de un mayor número de técnicas ha suscitado cambios más rápidos.

Los LMAN forman una corriente de investigación cuya evolución histórica ha mostrado de forma muy natural la automatización del desarrollo de "software". Con los LMAN es posible escribir programas en un nivel de abstracción mayor que los que se puede lograr con los lenguajes convencionales de alto nivel. Las pretensiones son que el programador describa las estructuras de control, las estructuras de datos y las operaciones en el ambiente natural en que suceden las aplicaciones y/o el dominio del problema. Para alcanzar el objetivo, al programador se le proporcionan un conjunto limitado de constructores del lenguaje que están orientados al dominio del problema, evitando de esta forma los detalles que se relacionan con la computación que no son esenciales al problema. De esta manera, la tarea de solución se simplifica teniendo al lenguaje como una guía.

Con los LMAN el usuario no tiene que tratar con características generales, él puede evitar muchas representaciones que se derivan de la necesidad de decidir cómo un algoritmo se expresa en relación con los procedimiento generales. Sin embargo, la simplicidad y la facilidad que los LMAN dan en la programación es un potencial que se compromete con:

a Una reducción en la flexibilidad para el programador.

b) Algunas restricciones en la elección de representaciones

c Una reducción en la eficiencia en la ejecución.

En otro orden de ideas, los SBC (Sistemas Basados en Conocimiento) son una metodología que necesita adquirir del usuario las especificaciones de los requerimientos del programa que se desea construir. Las especificaciones, con frecuencia, se expresan en lenguaje natural restringido utilizando la terminología que se tiene en el dominio del problema, con la finalidad de facilitar los medios de expresión. En algunos casos, los SBC pueden tratar con especificaciones incompletas o inconsistentes, que con el uso de conocimiento en el área de aplicación se pretende resolver las ambigüedades para interpretar adecuadamente las especificaciones; [MANN75-79], [GREE75-79], [BARS76-79C] y [KANT77-81].

Los SBC se fundamentan en la transformación de las especificaciones a un sistema de procesos que tienen asociados objetos de datos. Para afectuar esas transformaciones el sistema deberá disfrutar de una pericia parecida a la empleada por los analistas de sistemas, en el diseño de los sistemas de "software".

2.2 METODOLOGIA EN LENGUAJES DE MUY ALTO NIVEL.

En el desarrollo de sistemas de muy alto nivel el objetivo es el de proporcionar al lenguaje constructores de abstracción procedural y de datos, con la finalidad que el usuario escriba enunciados del programa en un sentido más descriptivo. En abstracción procedural, se incluyen inicialmente subrutinas y procedimientos, con construcciones posteriores para ciclos y bifurcaciones. Por otro lado, la abstracción de datos incluye estructuras de alto orden y sus operadores agregados. Como consecuencia los aspectos que caracterizan a los LMAN son fundamentalmente dos: abstracción procedural y abstracción de datos. Considerando que algunos sistemas incluyen módulos para tener referencia asociativa de los datos [WEGN78].

2.2.1 Abstracción Procedural.

Aunque ciertos lenguajes de programación convencionales admiten alguna especificación implícita del control flujo, en general, los programas convencionales demandan que se describa con exactitud el control de flujo, precisando en la codificación las etapas que se suceden en el algoritmo. Por ejemplo, Fortran admite que el programador escriba una expresión algebráica en lugar de una secuencia precisa de operaciones aritméticas, que son necesarias para calcular la expresión. Sin embargo, si planteamos una expresión matricial, no es posible resolverla con los objetos de datos matriciales y las operaciones asociadas, situación que sí es aceptada en APL [PERL75] y [GUIB78].

En los lenguajes de programación denominados de alto nivel, el usuario deberá dar los procedimientos etapa por etapa hasta llegar a la formación total del programa. En él, se muestra de forma explícita el control de flujo cuidando de las ramificaciones y ciclos. La ventaja en los LMAN es que implícitamente se ensamblan partes de código que corresponde a una labor del programador. De este último, solamente demanda la secuencia explícita de la información en esas partes del programa en donde es necesario detallar los controles. Para el resto del programa los LMAN deberá deducir el orden de proceso a partir de la información dada por el programador.

Una forma de especificación implícita del control de flujo es la que se puede observar en lenguajes aplicativos o funcionales. En ellos se describen las salidas en términos de sus

entradas, evitando la liga de variables y transferencias. En un programa funcional el orden en que aparecen las proposiciones suele ser irrelevante y, la propiedad de "asignación simple" (i.e. ninguna variable puede ser asignada a un valor por mas que una proposición) conduce a que resultados intermedios sean nombrados. Esta característica ha sido usada en varios lenguajes de simulación ofreciendo al programador un modelo de descripción funcional del problema que se desea simular.

Otra forma de especificación implícita del control se manifiesta con la característica de permitir ensamblar partes del lenguaje para que sean ejecutadas concurrentemente. En un sistema de procesador simple, esto significa ceder la responsabilidad de decidir el orden en que las partes son ejecutadas. Una versión de esta capacidad se expresa con la construcción de un lenguaje especial que indique que un conjunto de proposiciones puedan ser ejecutadas en paralelo. En un programa de flujo de datos se ofrece la posibilidad de exponer paralelismo, no presentando restricciones en la secuencia que se tiene por la dependencia en los datos. Los Lenguajes de Flujo de Datos [CHAP86], toma el concepto fundamental de funcional o de aplicación de operadores, ya que produce valores que son usados por otros operadores. En ellos no es necesario considerar un "program counter" de instrucciones que con valores de entrada se ejecute y produzca un conjunto de valores de salida que demandan otras instrucciones.

2.2.2 Abstracción de Datos

Un concepto importante usado en el desarrollo de lenguajes de programación es el de definir estructuras de datos que permitan facilidades al programador en descripciones más naturales con menos detalle. Por ejemplo: arreglos multidimensionales, estructuras de datos heterogéneas (registros) y estructuras anidadas, entre otras. En los LMAN se manejan estructuras de datos de un nivel de abstracción mayor que no están directamente orientadas a la máquina, liberando al usuario de muchos detalles. Además de que se produce código trasportable independiente de una máquina especifica. Lo esencial es que las estructuras de datos ocurran de manera natural en el ambiente de los problemas permitiendo conceptualizar, organizar y expresar las mencionadas directamente en los programas.

Los LMAN permiten al programador emplear estructuras de alto nivel (conjunto, multi-conjunto, relaciones y patrones,entre otros.), asociándolas a operadores agregados que pueden ser aplicados directamente a una estructura completa. De esta forma tenemos un aspecto de representación y otro de comportamiento para dar el concepto de Abstracción de Datos o Tipo de Datos Abstracto (TDA).

Desde la década de los 50's, con Fortran se vió la conveniencia de identificar los valores, que obedecía en parte, a la representación en "hardware", y en parte, al cálculo computacio-

nal relacionado con los códigos de operación. De esta forma nace el concepto de tipo, como la información que sirve al compilador para la determinación de valores durante el curso de la computación. Los tipos sirven también para detectar postulados ilegales y para determinar qué operaciones primitivas son legales o permitidas sobre cierta variable, digamos el compilador se fija en su tipo. Los tipos en un lenguaje dividen el conjunto de los objetos en subconjuntos caracterizados por la estructura o por la aproximación de comportamiento algebráico, definiendo una equivalencia para formar una clase.

Las metodologías en programación han extendido el concepto de tipo a nuevas formas de abstracción. En un TDA se requiere encapsular información para suministrar una representación de un objeto de dato complejo, considerando la implementación de la estructura y las operaciones que el objeto es capaz de manejar. La representación interna del objeto mostrado por un TDA es completamente escondida, solo las operaciones para el manejo de la representación están disponibles al usuario. De esta forma los TDA, tienen asociada una clase de abstracción que se maneja con mecanismos de llamados de procedimiento y secuencias de protocolos que emplean los programadores.

La noción de TDA aparece por vez primera en 1974 con el trabajo de Liskov [LISK74]. En él, se muestra un modelo para encapsular objetos abstractos y dá pie a la investigación de lenguajes con mecanismo de encapsulamiento. Alphard [WULF76] y CLU [LISK77] son dos lenguajes que usan estas ideas con dos enfoques diferentes para la asignación y creación de objetos. Por un lado, Alphard maneja el concepto de forma "form" -se deriva del concepto de clase de Simula [DAHL72]-, en donde se encapsulan las descripciones asociadas a los datos y un conjunto de funciones relativas. Por otro lado, en CLU se construyen módulos en términos de agupamientos "cluster" para la implementación de la abstracción.

La noción de tipo determinado por el modo de comportamiento a la aplicación de las operaciones, no siempre, es compatible con la idea de mecanismo de acceso uniforme para los componentes de un objeto agregado. Como un ejemplo, considere los números complejos en su representación polar y en reales. Para convertir el tipo polar de números complejos en datos reales, es necesario tener definidas operaciones para el tipo y una estructura de datos transparente en una definición de encapsulamiento. El acceso se hace solo a operaciones en la estructura de datos, aunque, la operación afecte a cada uno de ellos. Es por esta razón que los tipos determinan que operaciones son legales en un lenguaje.

En un flujograma los tipos de datos que principalmente fluyen por los arcos son relaciones conformadas con agregaciones de atributos. Dichos atributos tienen asociado diferentes tipos que son declarados por el usuario y guardada su especificación dentro del descriptor. De esta forma LIDA maneja relaciones y los operadores que aparecen en el nivel conceptual respectivo,

manejados internamente en un nivel de desagregación de atributos y valores, cuya tipificación se encuentra declarada en el Descriptor de Archivos. Para más detalles ver el capitulo 4 (El Modelo de Datos de LIDA).

2.2. Referencia Asociativa

En algunos lenguajes de programación que proporcionan estructuras de datos con niveles agregados son provistos por su Referencia Asociativa. El objetivo de la referencia asociativa es que el programador pueda identificar y extraer los objetos agregados con los que trabajan las estructuras. Es posible proporcionar una descripción parcial de ellos y expresarla en términos de sus valores o sus propiedades. Por ejemplo, considere la representación matricial con estructuras de datos de arreglos; en donde es necesario implementar los operadores asociados al álgebra matricial. Los constructores son identificados dentro de un dominio en el cual las operaciones están bien definidas. Los operadores agregados se asocian referencialmente a operaciones de subclase u otros dominios.

En algunos casos, las operaciones agregadas recurren a expresiones de alto nivel en donde se tienen iteraciones que admiten ciclos, pero que pueden ser expresados simplemente por declaración. La referencia asociativa mencionada corresponde a referencia vectorial o a los detalles de valores sobre números de punto flotante.

2.3 CLASIFICACION DE LENGUAJES DE MUY ALTO NIVEL.

Las áreas más prominentes de aplicación para los LMAN son las que tienen un dominio amplio y bien definido del problema [SAMM72]. Dichas áreas deben ser lo suficientemente vastas para requerir de un lenguaje de programación y, lo suficientemente coherentes para admitir una semántica de dominio que permita ser capturada con el diseño de un lenguaje. Los objetos de datos, operadores y estructuras de control, deben ser los más naturales y adecuados dentro del dominio de aplicación, para que de esta manera, tanto el que escribe como el que lee un programa en un LMAN, tenga un marco de referencia conceptual dentro del área de conocimiento respectiva.

Un LMAN de propósito general es aquel que intenta tener aplicación en un rango amplio de problemas de programación, de la misma forma, que lo tienen los lenguajes convencionales de propósito general. Sin embargo, se considera que algunos son más adaptables a ciertas clases de aplicaciones que otras. Podemos mencionar que SETL (Set Theory Language) y APL son ejemplos apropiados de LMAN de propósitos generales [SCHW73]. SETL con una sintáxis y semántica de teoría de conjuntos permite programar algoritmos rápida y sucintamente, no se requieren declaraciones en estructura de datos y admite objetos formalizables bajo la

teoria de conjuntos finitos con sus operaciones asociadas. Algunas operaciones de la teoria de conjuntos son extendibles dinámicamente a longitudes arbitrarias, como son: la concatenación de eneadas, la recuperación indexada y el almacenamiento de componentes. Finalmente, en SETL las estructuras de control se conservan con: bifurcación, ciclos, selección e iteraciones sobre conjuntos, incluyendo los cuantificadores universal y existencial.

Diversas necesidades que se presentan cuando se han desarrollado investigaciones en IA ha propiciado la creación de nuevos sistemas de programación. Nuevos lenguajes han incorporádo facilidades de programación que incluyen aspectos inovadores:

Nuevos tipos de datos y mecanismos de acceso para almacenar expresiones.

Estructuras de control más flexibles que incluyen procesos multiples y retrocesos.

* Comparación de patrones que admite comparaciones de elementos de datos con una plantilla.

* Mecanismos de deducción automática que pueden modificar las bases de datos y decidir cuales subrutinas deben correr en una situación dada

El incorporar ciertas facilidades a los lenguajes para la IA, ha permitido crear LMAN que no solo tienen su enfoque en términos de usuarios finales; sino también, a expertos programadores del campo de IA. En el articulo "New Programming Language for Artificial Intelligence Research." [BOBR74], se encuentran detalles y una discusión amplia acerca de lenguajes de IA como son: SAIL, PLANNER, CONNIVER, QA4, POP-2, POPLER, entre otros.

Durante mucho tiempo la investigación en el campo de interacción hombre-máquina ha sido el quid del desarrollo de "software" para la administración y la automatización de las oficinas. El área particular de aplicación denominada procesamiento de datos, ha suministrado un vasto ambiente para el desarrollo de sistemas y lenguajes que intentan generar automáticamente programas que resuelvan problemas a la medida. El desarrollo de los LMAN para la administración y automatización de oficinas han estado sujetos al compromiso entre la generalidad de sus aplicaciones y la simplicidad en el uso del lenguaje; aspecto que motiva nuevas formas ergonómicas de interacción hombre-máquina. El enfoque de programación visual es una fructífera tendencia que facilita la transición entre los diseños diagramáticos en ingeniería de "software" y la solución de los problemas en la generación de programas (capítulo 3).

Podemos mencionar que RPG es el precursor de LMAN dentro del campo del procesamiento de datos; a partir de éste, algunos otros han sido creados con la finalidad de ampliar su alcance y mejorar

la potencialidad. Ejemplos de ellos podemos mencionar a: "CONVERT: A High Level Translation Definition Language for Data Conversion" [SHU75]. SSL [RUTH76]. " BDL: Business Definition Language" desarrollado en el Centro de Investigación de la IBM Thomas J. Watson [HAMM77] y [ELLI80]. "SBA: System for Business Automation" [ZLOO77a]. Lenguaje Model II y generador de programas asociado [PRYW79]. "Forms Processing Language" y "Business Procedure Definition Language" [SHU82]; "FORMAL: A Forms-Oriented, Visual-Directed Application Development System" [SHU85], desarrollado en el Centro Científico de los Angeles de la IBM.

2.3.1 Sistema Business Definition Language.

Business Definition Language [HAMM77] y [ELLI80] tiene como acróstico BDL. Es un lenguaje de muy alto nivel enfocado al procesamiento de datos con los principios de modularidad y programación estructurada. BDL puede ser usado como lenguaje de programación para programadores no-expertos, para que describan y construyan sistemas de problemas particulares; o también, para ser usado como "lenguaje intermedio" de un sistema de Programación Automática, cuyo frente adquiere conocimiento del dominio del problema, construye una estrategia de solución y lo expresa en BDL. El sistema maneja conceptos y estructuras orientadas al problema para que los especialistas en el área las usen en programas fácilmente leibles y modificables.

El sistema BDL consiste de tres componentes principales que describen la naturaleza e interrelación de objetos de documentos, etapas, trayectorias y archivos:

1. Componente de Definición de Formas "Form Definition Component" (FDC).

2. Componente de Flujo de Componente "Document Flow Component" (DFC).

3. Componente de Transformación de Documentos "Document Transformation Component" (DTC).

La parte FDC es un subsistema de interacción gráfica que ayuda a los administradores a definir las formas por medio de rectangulos y llenado de contenidos de campos sencillos. Una forma en BDL, es comparable a la noción de forma en blanco de Officetalk (capitulo 3). En ella primero se tiene un trazo físico de la forma en donde se especifica su tamaño con la información de preimpresión, campos y encabezados. Los detalles de la información de la forma se definen usando, también FDC, por medio de la especificación de los nombres de campos, tipos de datos, formatos de datos, nombres de los grupos de campos, y campos llaves para ordenar grupos de campos.

El Componente de Flujo de Documentos describe el flujo de datos por medio de una gráfica dirigida; los elementos de la gráfica son etapas y archivos (nodos) interconectados por arcos

que definen trayectorias. El conjunto de nodos en una gráfica de DFC se construye con rectangulos que representan etapas y circulos que representan archivos. Los arcos de lineas solidas dirigidas interconectan etapas y los arcos de lineas punteadas interconectan etapas con archivos. Cada arco se etiqueta para definir el tipo de documento que fluye sobre la trayectoria correspondiente.

Por medio del sistema gráfico FDC es posible construir gráficas jerárquicas (figura 2.2), en las cuales cada nivel intermedio en la jerarquia se construye de uno o mas nodos compuestos. Un programa en BDL se define primero por medio de la especificación de una gráfica compuesta de etapas, trayectorias y archivos, que ilustran las unidades organizacionales de la administración y el flujo de documentos entre esas unidades. Los refinamientos son hechos de arriba-hacia-abajo, descomponiendo las etapas en etapas constitutivas al nivel de irreducibilidad. Una etapa de automatización es refinada en una etapa irreducible con una función para definirse en la componente DTC.

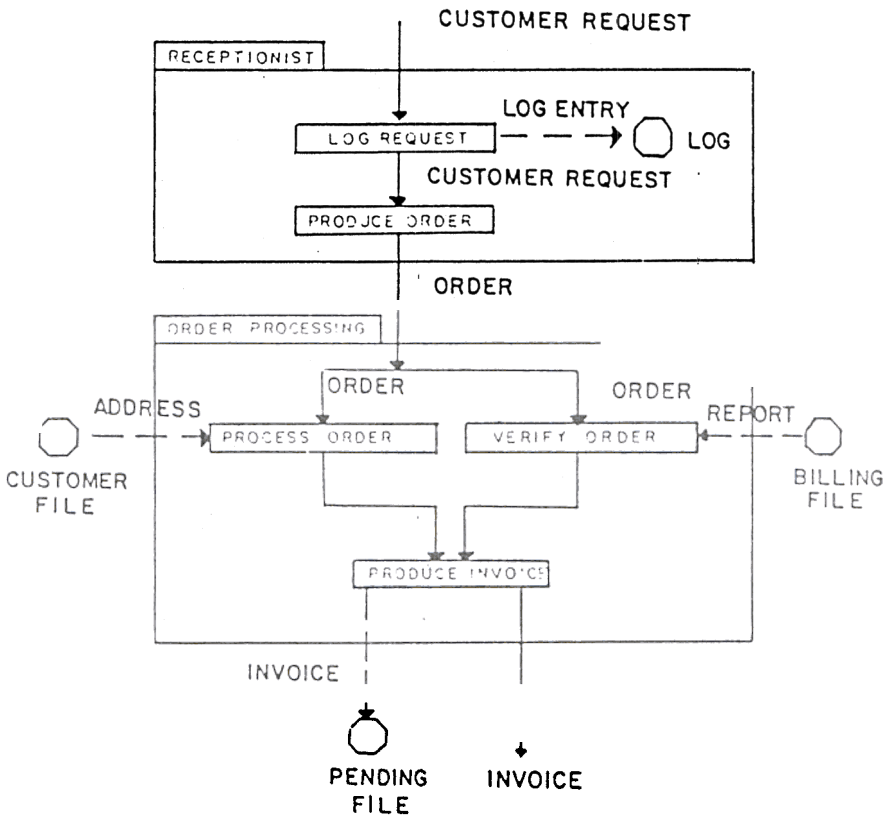


Fig. 2.2 PROCESAMIENTO DE ORDENES EN F D C

El Componente de Transformación de Documentos (DTC) es un lenguaje de programación arbitrario, en donde cada procedimiento es invocado cuando una ejecución de DFC está disponible con documentos de entrada. En BDL, la componente DTC es un LMAN con enfoque al procesamiento de datos agregados. EL DTC contiene un marco referencial algoritmico que le permite construir en cada etapa programas. El programador usa el marco de referencia para definir transformaciones particulares de la información a partir de un documento de entrada en otro de salida. La etapa de interpretación deberá expresarse para cada valor del campo en un documento de salida; considerando que las expresiones deberán ser expresadas en terminos de operadores aritméticos ordinarios, expresiones lógicas y expresiones relacionales (ejemplo en la figura 2.3).

NOMBRE DEL GRUPO DE CAMPOS	CASUALIDAD/ DERIVACION	NOMBRE	DEFINICION
1 Factura	ONE PER Nota	Nota	INPUT
2 Nombre	Nombre Remesa	Nombre Remesa	IN Nota CAUSE OF Factura
2 Dirección	Dirección Remesa	Dirección Remesa	IN Nota CAUSE OF Factura
2 Productos	ONE PER en Productos	Productos Nota	IN Nota CAUSE OF Factura
3 # Producto	En # Producto	En # Producto	IN En Producto CAUSE OF Producto
3 Precio	Precio	En Precio En Producto	IN Producto CAUSE OF Producto
3 Cantidad	En Cantidad	En Cantidad En Producto	IN En Producto CAUSE of Producto
3 Extra	Precio x Cantidad		
2 Subtotal	SUM (Extra)		
2 Descuento	9% Clase="A" 7% Clase="B" 3% Clase="C"	Clase Nota	IN Nota CAUSE OF Factura
2 Descuento	Subtotal x Descuento		
2 Total	Subtotal - Descuento		

Figura 2.3 Programa en BDL para el Procesamiento de Ordenes.

2.3.2 Lenguaje MODEL II.

Basandose en los trabajos en generación automática de programas para la conversión de datos [RAMI73] y extensiones conceptuales a programas con procesamiento de transacciones [RIN76]; Prywes, Pnueli y Shastry desarrollarón Model II [PRYW79]. El sistema contiene un lenguaje de no-procedimiento con la habilidad de manejar problemas complejos en: la entrada y la salida; base de datos; y el reporte de requerimientos. Ofreciendo una simplificación en la programación por parte del usuario.

El lenguaje MODEL II y el generador de programas asociado se basa en la construcción de programas que no están formados de procedimientos, solo se declaran propósiciones que se dan en cualquier orden facilitando al usuario el control. Además no exisisten constructores de control de iteraciones, entrada/salida y localizaciones de memoria. La tarea de ordenar las proposiciones para la ejecución y proporcionar los comandos de control se llevan al cabo por el generador automático de programas.

En Model II el usuario provee en una primera etapa los requerimientos de procesamiento de los datos, para posteriormente interpretar los requerimientos que constituirán un programa de especificación, consistente en tres partes principales:

1) El encabezado que consiste del nombre de los programas y de las bases de datos.

2) La descripción de los datos, que consiste de la estructura de la fuente y los datos objeto incluidos en la especificación.

3) Las aseveraciones, como parte relacional que especifica las relaciones entre la fuente y las variables objeto.

Cuando todos los problemas son resueltos el sistema produce un programa en PL/I o Cobol.

El principio del lenguaje Model II fué el de evitar que los usuarios requirieran de un alto manejo de programación. A este respecto el usuario compone sus necesidades de especificación concentrandose solamente en la descripción de datos y relaciones. Para facilitar el uso de base de datos y reportes el lenguaje muestra facilidades para describir un esquema arbitrario de datos, definido en estructuras jerárquicas como sucede en PL/I y COBOL. Debido a que las iteraciones en un programa es un concepto de programación convencional, la otra caracteristica que se impuso a MODEL II fué el de iteraciones implícitas.

El procesador Model II infiere la información necesaria para construir sus controles de repetición; considerando las especificaciones de repetición para datos y variables subscritas en aserciones. Finalmente, el otro aspecto relevante es la interacción sistema-usuario en el manejo matemático de la incompletitud, ambigüedad e inconsistencia de las especificaciones, aspectos

todos relacionados con la verificación en programación automática.

De los lenguajes y sistemas antes referidos se tienen puntos en común, que en algunos casos se muestra mayor relevancia de acuerdo a los principios de diseño para los cuales fueron diseñados. Entre los aspectos relevantes podemos concluir:

* El uso de especificaciones del flujo de datos que describen la estructura total y la organización del sistema.

* La especificación de módulos del sistema. Usualmente las salidas se describen funcionalmente a partir de las entradas.

* El uso de base de datos, en donde los archivos son accesados asociativamente.

La importancia del último punto ha originado una clase propia de sistemas denominados Sistemas Manejadores de Bases de Datos (SMBD). Ellos resuelven problemas de organización y transformación de los datos, así como, el manejo y la recuperación de la información de las estructuras de almacenamiento. En los SMBD se han incluido lenguajes de muy alto nivel para que el usuario pueda manejar los datos, hacer consultas y desarrollar aplicaciones. Los LMAN son un vehículo de acceso al sistema de cómputo, en donde, se tiene un proceso de transformación desde un nivel conceptual alto hasta niveles relacionados con la estructura de la máquina como se refiere en la figura de programación automática. Cada proceso debe cuidar de mantener la validez semántica y la integridad de los datos, con transformaciones que reconozcan las estructuras y contemplen la compatibilidad de los objetos de datos del modelo. Con lo que se refiere a esta última clase de LMAN recomendamos el articulo [JARK85], en donde se muestra un panorama en la investigación del desarrollo de Lenguajes de Consulta, con sus niveles taxonómicos y evaluaciones.

Finalmente, concluimos que PA es una área de la IA importante para el desarrollo automático de "software", cuyo objetivo ambicioso de solucionar problemas a partir del planteamiento de los mismos, ha propiciado crear una tecnología con herramientas de niveles muy diversos de complejidad y distintas capacidades. De este capítulo se desprende que los LMAN son piedra angular para la solución de problemas en base a la generación de código, con una tendencia natural hacia el análisis y diseño automático de los sistemas.

C A P I T U L O 3

FILOSOFIA Y EVOLUCION DE LOS LENGUAJES GRAFICOS

El objetivo de este capítulo es el de dar el marco teórico de la programación visual con la finalidad de ubicar y evaluar a LIDA. Presentamos cómo las imágenes gráficas proporcionan un recurso para la programación, siendo los diagramas los paradigmas para la creación de lenguajes visuales. Para la definición de requerimientos, los lenguajes visuales han mostrados ventajas para expresar, entender y documentar problemas, por tal razón los tratamos en la sección 3.4. En la sección 3.5 y 3.6 se dá un panorama de diversos sistemas con diferentes enfoques, debido a que programación visual en general usa todo un entorno gráfico en el diseño, la demostración y la documentación;

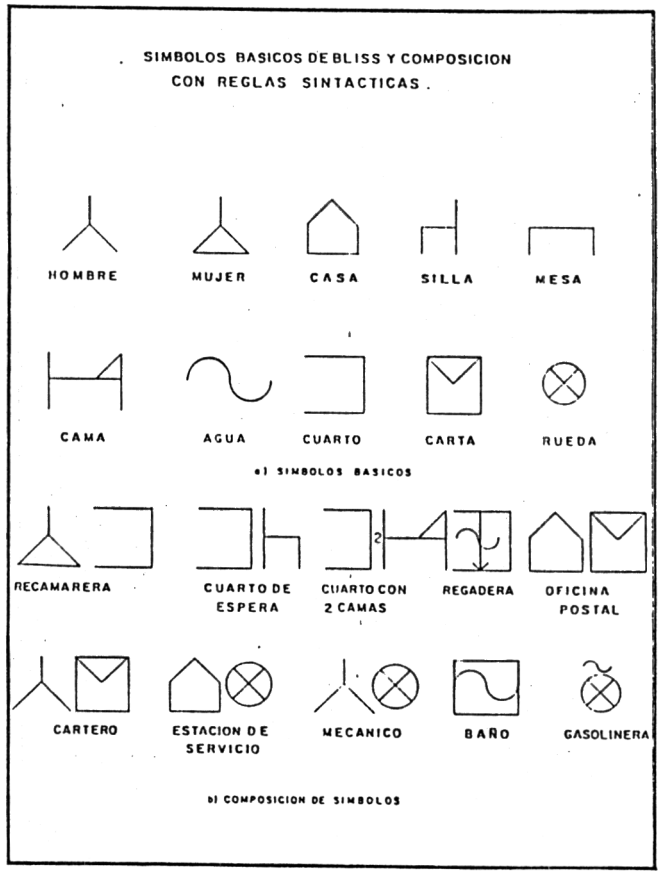
La comunicación mediante imágenes ha incursionado en diversas disciplinas de conocimiento, debido a que son una herramienta cognositiva útil y poderosa. En particular, para la administración y la computación se han creado elementos pictóricos con la finalidad de cumplir con ciertos requerimientos de enfoque [PURC81], [ROSS77] y [SHU85]. Por otro lado, varios grupos de trabajo en el área de IA interesadas en la interacción hombre-máquina estan buscando nuevas alternativas de tipo visual e icónicas para la solución de problemas ergonómicos y funcionales [FOOG84] y [JARK85].

3.1 EL USO DE GRAFICAS EN PROGRAMACION

Desde la década de los cincuentas en el campo de la estadística se mostró gran interés en la representación gráfica de datos, considerandose tanto sus aspectos descriptivos como analíticos. Sin embargo, no es hasta la década de los setentas -catalizado por el desarrollo tecnológico de la computación- cuando las gráficas se presentan como una disciplina intelectual; aspecto que se vio reflejado con la publicación de varios artículos referentes al área: [BENI78], [FIEN77], [KRUS77], [CHER73], [DREY72], [ROYS70] y [SCHM54].

Los símbolos pictóricos, gráficas y diagramas son manifestaciones visuales que tienen por objetivo: representar, ilustrar y comunicar objetos e ideas. Los datos como objetos pueden ser presentados y descritos en gráficas, mostrandose en estas mismas nuevas ideas obtenidas de las relaciones funcionales que se describen en la gráfica.

Por un lado, las ideas concebidas por el comunicador son sintetizadas en diagramas y símbolos gráficos -llamados Iconos- ofreciendo una atención atractiva al lector que le facilita el recordar conceptos y relaciones. Por otra parte, el concepto de


SIMBOLOS BASICOS DE BLISS Y COMPOSICION
CON REGLAS SINTACTICAS.
HOMBRE
MUJER
CASA
SILLA
MESA
CAMA
AGUA
CUARTO
CARTA
RUEDA
a) SIMBOLOS BASICOS
RECAMARERA
CUARTO DE ESPERA
CUARTO CON 2 CAMAS
REGADERA
OFICINA POSTAL
CARTERO
ESTACION DE SERVICIO
MECANICO
BAÑO
GASOLINERA
b) COMPOSICION DE SIMBOLOS

En el diseño de lenguajes gráficos se requiere de la definición de símbolos que dependen de las necesidades que se pretenden satisfacer, relacionadas con su significado bajo la siguiente clasificación:

* DE REPRESENTACION.
* ARBITRARIOS.
* ABSTRACTOS.

Los símbolos de representación reproducen adecuadamente fiel y simplificada la imágen de un objeto o una acción. Los símbolos arbitrarios son aquellos que se crean para que tengan que ser aprendidos. Y finalmente, los símbolos abstractos reducen elementos de un mensaje a términos gráficos; los cuales pueden ser representativos, pero también, haberse simplificado por el diseñador o cambiado al transcurrir el tiempo.

En la Figura 3.1 presento algunos iconos o símbolos básicos de Bliss. Algunas combinaciones entre símbolos básicos se constituyen cómo reglas sintácticas para construir sentencias visuales compuestas [DREY72].

3.1.2 Gráficas.

Las gráficas son una herramienta cognositiva que resúmen una gran cantidad de información para que el lector de una sola vista examine los datos, los analíse y determine las relaciones entre ellos. Por medio de una gráfica los usuarios pueden examinar y manejar objetos de una base de datos sin conocer mucho acerca de ellos. En este caso podemos señalar posiciones espaciales y con el uso de iconos podemos convenir cierta información que de alguna otra forma sería más difícil de manejar y menos objetiva en su presentación. Una clasificación es la siguiente:

a) Gráficas analíticas que describen relaciones teóricas tales como: funciones de probabilidad, funciones de densidad y funciones de riesgo.

b) Las gráficas no numéricas como suelen ser los: nomogramas, caras de Chernoff, tablas; y especialmente las gráficas con escala de tipo nominal.

c) Gráficas que despliegan datos y resultados de análisis, por ejemplo: diagramas de series de tiempo, histogramas obtenidos por el método de Monte Carlo.

3.1.3 Diagramas

Los diagramas son dibujos geométricos que tienen como objetivo demostrar una proposición, resolver un problema o expresar gráficamente la ley de variación de un fenómeno. Los diagramas son esquemas de excelente presentación visual en donde intervienen texto, números e iconos, con el objetivo de ilustrar y comunicar ideas. Como ejemplos podemos mencionar: diagramas trifásicos, diagramas de Fox, diagramas de Venn, diagramas de

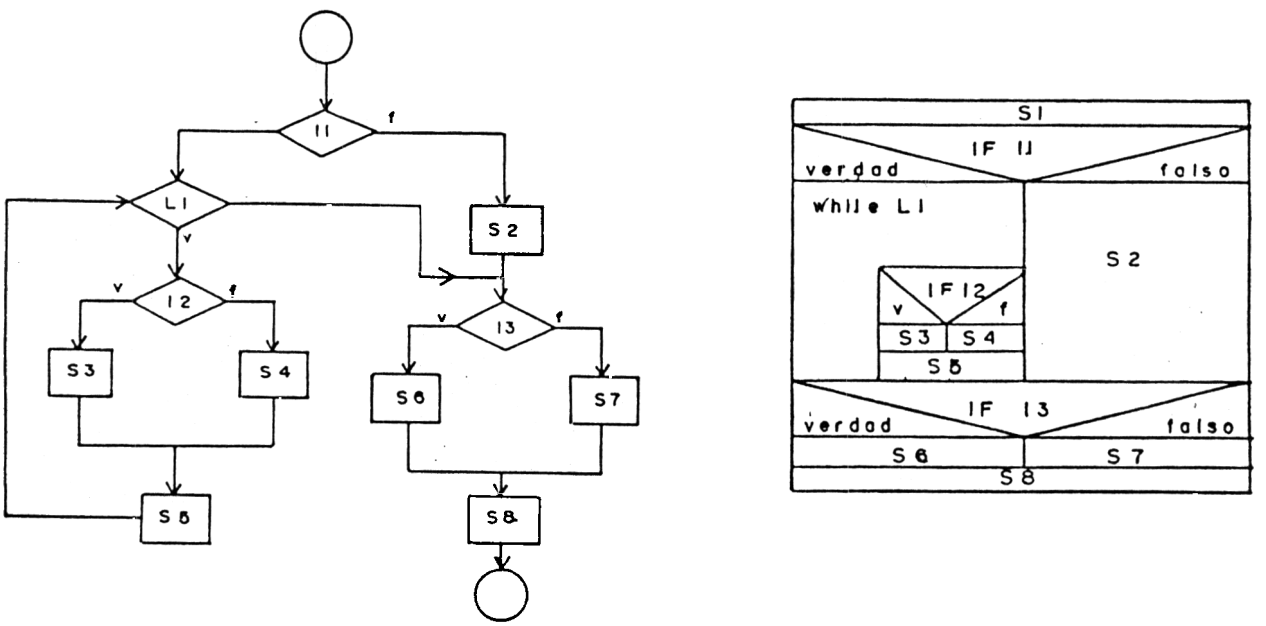
control de flujo, diagramas de flujo de datos, diagramas de Entidad-Relación y diagramas de Warnier, entre otros

3.2 DIAGRAMAS DE CONTROL DE FLUJO .VS. FLUJO DE DATOS.

3.2.1 Diagramas de Control de Flujo.

La forma gráfica más tradicional en computación es la de control de flujo o diagramas de flujo (Fig. 3.2 (a)). Los diagramas de flujo se desarrollaron como una herramienta para los programadores, con la finalidad de clarificar con gráficas la intrincada estructura de control, mientras deja por un lado, la declaración de tipos y estructuras de datos. Los diagramas de flujo fueron ampliamente usados en la elaboración de programas en donde se detalla minuciosamente el procedimiento, como suele ser en programación de lenguajes de ensamblador y Fortran. A pesar de que esta técnica ha caído en desuso, la ayuda gráfica resuelve algunos problemas de entendimiento de como trabaja el programa. Algunos lenguajes icónicos de programación con filosofía de demostración, por ejemplo PECAN [REIS84], tienen la posibilidad de mostrar una vista de estos diagramas; o bién, los de Nassi-Shneiderman [NASS73] que en seguida se explican.

Los diagramas de Nassi-Shneiderman llamadas también cartas estructuradas (Fig. 3.2 (b)), son diagramas que imponen una estructura en el control del programa. En dichos diagramas no se muestran expresiones, llamados a procedimientos y tipos; considerando que la estructura de control no es tan clara como (Fig. 3.2 (a)).



(a) Diagramas de Flujo (b) Diagramas de Nassi-Shneiderman

Fig. 3.2 Diagramas de Control de Flujo.

3.2.2 Diagramas de Flujo de Datos.

Los diagramas de flujo de datos están constituidos por nodos que representan funciones y arcos que representan los datos que determinan las dependencias entre las funciones (Figura 3.3). En la gráfica de flujo de datos los valores estan representando "tokens" sobre los arcos que determinan el control de flujo.

La estructura gráfica del flujo de datos propícia un diseño completamente modular con vistas en módulos independientes cuya interconección se manifiesta a través de los arcos que pasan parámetros a cada uno de los nodos que tienen funciones específicas . Debido a que los diagramas de flujo de datos han mostrado ser muy útiles en la representación funcional y de proceso, su aplicación ha sido directa al análisis y diseño de sistemas. En ingenieria de "software" en la metodología de Gane/Sarson se emplean diagramas de flujo de datos para el modelado de procesos.

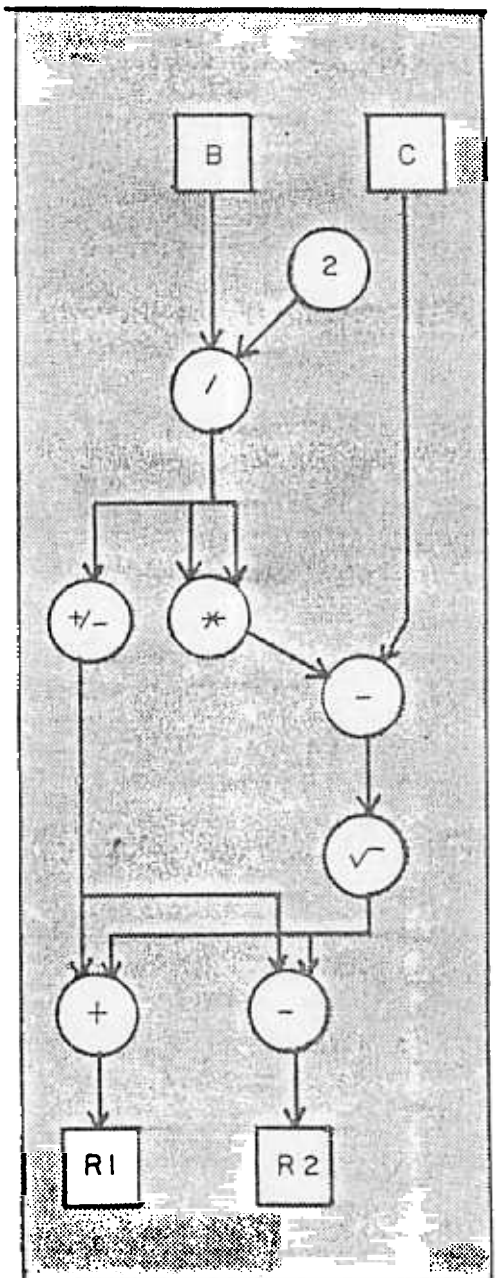


FIG. 3.3 DIAGRAMA DE FLUJO DE DATOS

3.3 LENGUAJES GRAFICOS DE FLUJO DE DATOS.

Los Lenguajes de Flujo de Datos (LFD) son lenguajes funcionales que tratan solamente con valores y no con nombres de los contenidos de los valores (i.e. direccciones); considerando que no se tienen construida la noción de actualización global de memoria, un operador produce un valor el cual es usado por otros operadores.

La propiedad de flujo de datos que mantienen dichos lenguajes aventaja en representaciones gráficas en la definición de programas. En los LFD, el control de flujo se manifiesta con el flujo de los datos, simplificando la estructura y propiciando un diseño modular; sin restricciones en la secuencia que se impone por la dependencia de los datos en los algoritmos. De esta manera, es posible exponer todo el paralelismo en un programa de flujo de datos. Los lenguajes de flujo de datos como lenguajes gráficos (Figura 3.4) surgen de forma natural por sus propiedades y ventajas:

1) Los lenguajes de flujo de datos como lenguajes gráficos se generan con gráficas dirigidas que dán el significado formal del programa. El control de flujo en la ejecución del programa esta determinada por los arcos dirigidos.

2) La secuencia de acciones en un programa escrito en un LFD se determina por las simples reglas de disparo de la disponibilidad de los datos. Cuando los argumentos de un nodo están disponibles, se dicen que el nodo está disparable. La función asociada al nodo puede ser ejecutada (disparada). Es decir, aplicar los argumentos que son absorbidos, para después del disparo enviar los resultados a otros nodos que requieren de dichos resultados; apreciandose que en un momento dado varios nodos pueden estar ejecutandose. Una descripción detallada es mostrada en el capítulo 7 con el método de ejecución de un flujograma.

Este comportamiento permite una representación con una gráfica dirigida, en donde cada nodo representa una función y cada arco un canal sobre el cual fluyen los datos.

3) Una característica importante de los lenguajes de flujo de datos es su refinamiento o jerarquización en los diagrámas. Ciertos nodos del programa pueden ser definidos para ser refinados en otra gráfica de programa.

4) En un programa de flujo de datos el orden de ejecución esta dado por la disponibilidad de los datos que dice quién se va ejecutar: cuando todos los datos de entrada estan presentes, un nodo puede ejecutarse o dispararse (ver capítulo 7).

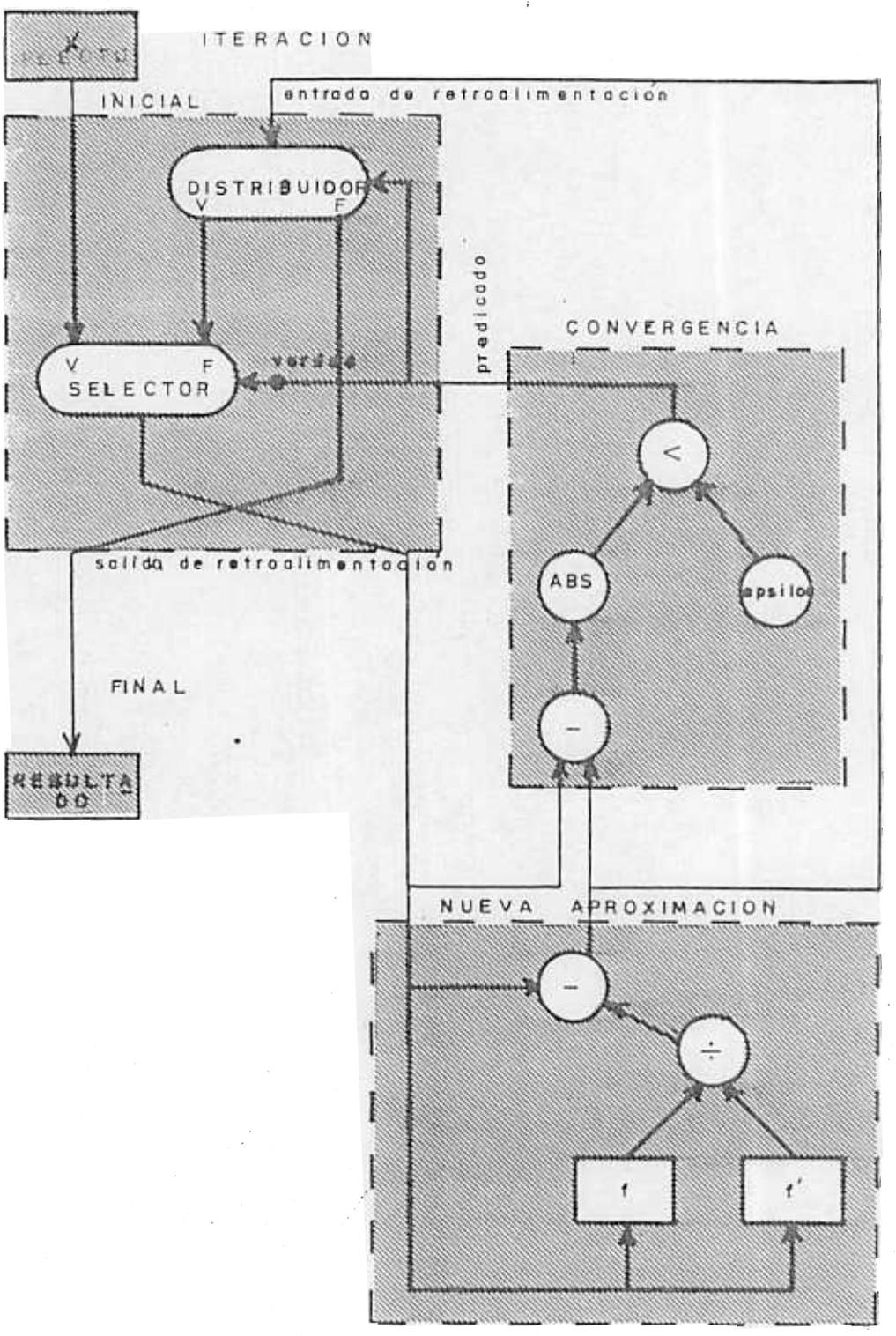


FIG. 3.4 PROGRAMA GRAFICO EN FLUJO DE DATOS PARA EL METODO DE NEWTON.

3.4 LENGUAJES GRAFICOS EN INGENIERIA DE "SOFTWARE".

En ingenieria de "software" una metodologia es una colección de métodos, que basados dentro de una filosofía común, se llevan al cabo en un marco de trabajo denominado ciclo de vida de los sistemas. Los métodos son procedimientos o técnicas que desarrollan alguna porción importante del ciclo de vida del "software" para: la definición de requerimientos, el diseño de la base de datos, el diseño de los programas,la codificación y el desarrollo de casos de prueba. Los métodos hacen uso de diagramas, formas y textos; con la finalidad de poder expresar, comunicar y documentar los problemas, para que el analista los resuelva escribiendo un programa.

En ingenieria de "software", el ciclo de vida de los sistemas se inician con la especificación de requerimientos, tomando a partir de este punto, la descripción de los datos abstractos y las acciones que deberán desarrollarse en la implantación del sistema. Dentro del contexto de especificación de requerimientos existen dos formas principales: los textos narrativos y la notación gráfica. A través de estas es posible expresar como los sistemas almacenan datos y procesan programas.

El texto narrativo reseña cada uno de los procesos de gestión requiriendo de mucho detalle; con la dificultad para el usuario de entender el enlace de las partes y la comprensión total del sistema. Con frecuencia, el modelo de textos narrativos tiene tropiezos con el establecimiento de su lógica, debido a que las formas de expresión que se establecen en los documentos suelen ser confusas o ambiguas. Sin embargo, una forma de resolver el problema, es el de reducir la variedad de formas a proposiciones lógicas simples [GANE79]. Dicha simplificación se logra en los documentos y procedimientos administrativos con frases imperativas y condiciones; permitiendo que el analista prepare un programa para un Sistema de Procesamiento de Datos.

En las formas gráficas, la determinación directa del significado de sus símbolos, obliga a los diseñadores y usuarios a ajustarse a la semántica para entenderse sin ambigüedades en el establecimiento de los requerimientos.

3.4.1 Lenguaje de Análisis Semántico.

El Lenguaje de Análisis Semántico fué desarrollado por Max Wilson [WILS75], en el Instituto "Information Automat" de la IBM. El análisis se lleva a cabo mediante conceptos básicos de: objeto, propiedad, relación, evento, procedimiento, decisión y conjunto. El análisis se describe en un diagrama de conceptos en el cual, cada concepto clave tiene asociado un símbolo gráfico, con flechas dirigidas que los relacionan. El resultado del análisis semántico de un problema y su ambiente se expresan por una serie de diagramas de concepto, con objetos, eventos, relaciones, etc.; que son identificados propiamente con los representados en el sistema de "software".

3.4.2 Diagramas de Flujo de Datos de Gane/Sarson

El Análisis Estructurado de Sistemas corresponde a la aproximación de Gane/Sarson [GANE79], que es una extensión de la metodología de diseño estructurado, que consiste en proponer un conjunto de herramientas y técnicas conceptuales para diseñar sistemas. El propósito del modelado lógico es el de partir de ideas vagas acerca de los requerimientos para convertirlos en definiciones precisas tan rápido como sea posible. El método propone una sucesión de trazos gráficos documentales de presentación de forma de flujo de datos y a un nivel lógico; trabajando en un proceso de refinamiento de la notación para analizar y entender sistemas de cualquier complejidad.

En los Diagramas de Flujo de Datos (DFD) se reconocen entidades externas como objetos que representan una fuente o destino de transacciones, por ejemplo: clientes, empleados, unidades administrativas, etc. Las entidades externas son simbolizadas por cuadros solidos, que puede ser identificado en la esquina superior izquierda. El flujo de datos se simboliza por flechas que representan en general, canales de flujo de información. En los DFD se tienen primitivos gráficos para fuente y destino de datos, representando el almacenamiento de los datos mediante cajas referenciadas. Los procesos son representados por cajas con bordes redondeados y texto interno identificador del proceso que transforma datos. La metodología de Gane/Sarson para el modelado lógico de un sistema, se establece fundamentalmente para el desarrollo de los sistemas con los DFD y el empleo del modelo de entidad-relación para el análisis de los datos almacenados.

En la figura 3.5 muestro un ejemplo de DFD de un sistema que representa la gestión de los prestadores de servicio en el departamento de Servicio Social.

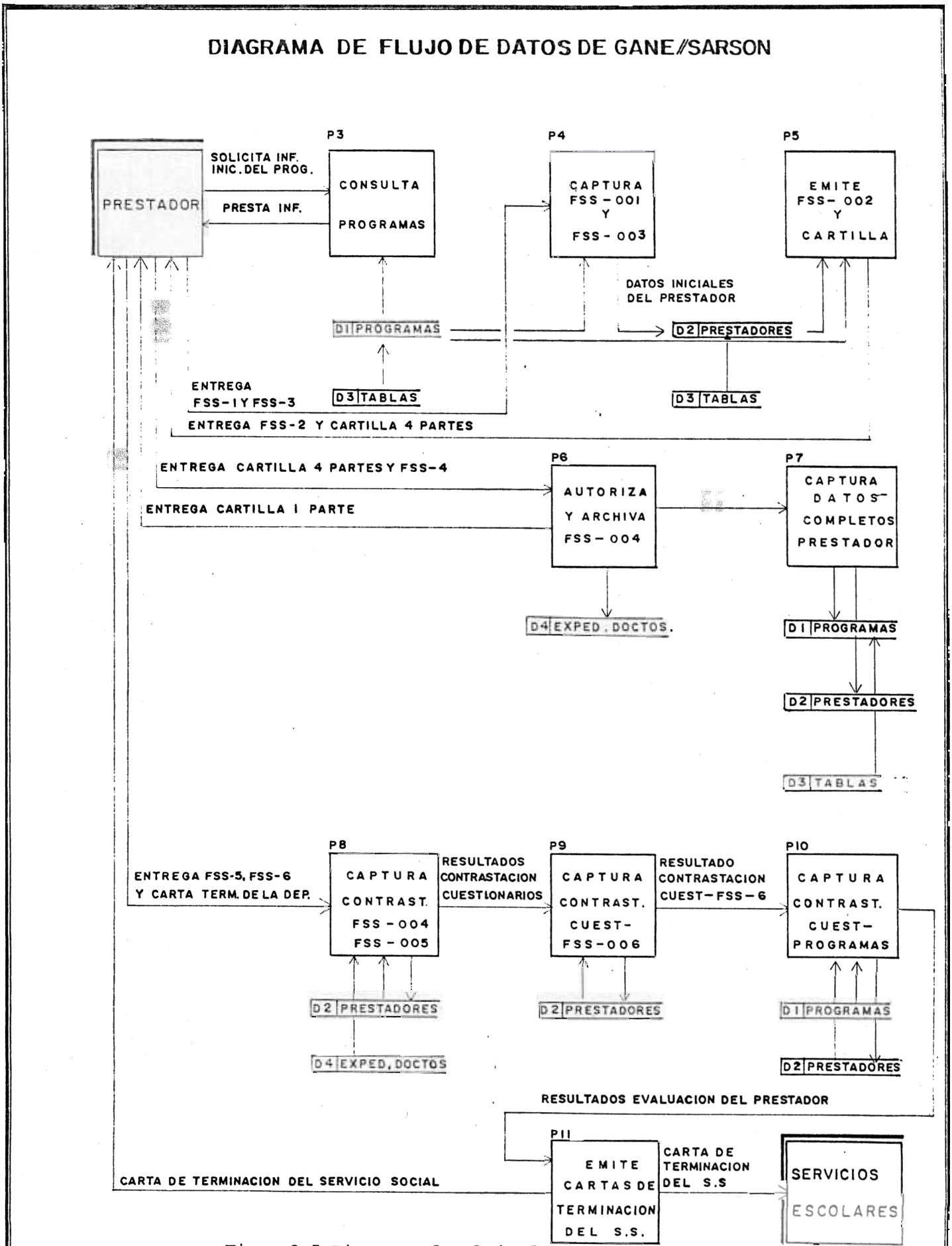


Fig. 3.5 Diagrama de Flujo de Datos de Gane y Sarson

3.4.3 SADT: Structured Analysis Desing Technic

La Técnica de Análisis y Diseño Estructurado (Structured Analysis Desing Technic -SADT-), es el método de definición de requerimientos basado en gráficas, que quizás, mas ha incursionado en el diseño de sistemas complejos [BRAC77]. El método publicado por Ross en 1977 en [ROSS77], ya había sido usado en sistemas administrativos de agencias financieras del gobierno de E.U.N. para entrenamiento militar [SOFT76]. Así como, para el diseño de "software" para sistemas de bases de datos muy grandes y sistemas de manufactura de teléfonos ayudada por computadora [AIR 74].

SADT consiste de dos partes principales:

1 Un lenguaje de diagramación de análisis estructurado consistente de cajas y flechas, lenguaje "Structured Analysis" (SA).

2) La técnica de diseño (Design Technic DT), que es la disciplina para establecer las ideas y desempeño que deberá ser aprendida y practicada con un buen empleo del lenguaje.

El lenguaje de SA es un lenguaje de comunicación de ideas que contiene un núcleo básico de primitivos pictóricos con los cuales se construye una estructura ordenada para documentar los sistemas de procesamiento de datos. A pesar de que SADT una de las mejores herramientas para el diseño, no ofrece posibilidades para la resolución de problemas.

El Lenguaje Gráfico SA, nucléo de SADT, parte de la idea clave de particionar en piezas mucho más simples el problema, caracterizado por la jerarquización de una descomposición de lo general a lo particular. Esta idea simple se denota en la notación del lenguaje gráfico mediante cajas de SA cuyos cuatro lados representan: entrada, control, salida y mecanismo; esquemátizados en la figura 3.6.

En una caja la entrada es transformada en una salida bajo un control y un mecanismo que lleva a cabo la transformación. La analogía se presenta como en una ecuación algebráica

$$Z = a*X + b.$$

En dicha ecuación X es el argumento de entrada; a y b son parámetros de control y finalmente Y es el resultado que sale. Los operadores * y + denotan el mecanismo que participan en la transformación.

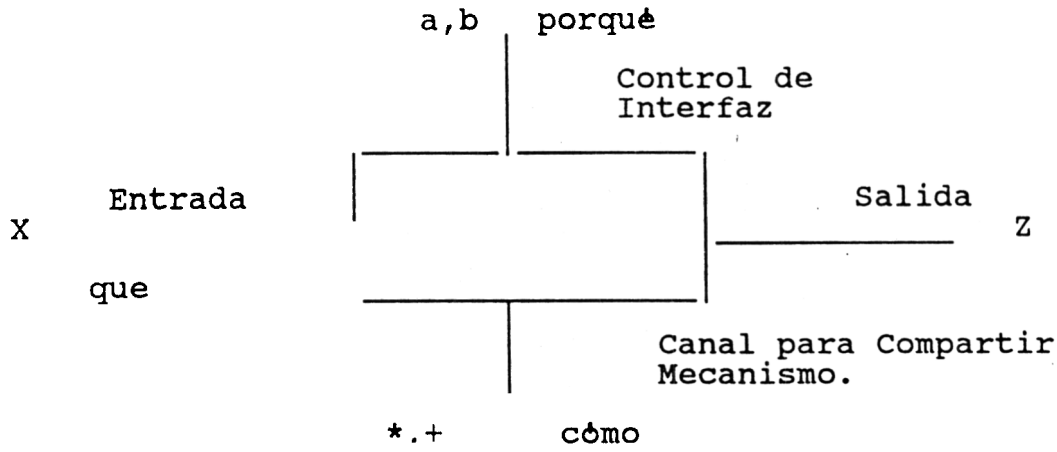


Fig. 3.6 Elemento Caja del Lenguaje SA.

En el lenguaje SA las interfaces que se tienen entre las partes son las entradas, el control y la salida para componer un todo que con una estructurara de flechas mantienen relaciones de restricción entre las cajas, sin representar el control de flujo. Las cajas son formas o transformaciones con la imposición de condiciones necesrias con nombres arbitrarios. Los segmentos de flechas son etiquetados con codigos de ICOM (entrada, control, salida y mecanismo). En SADT cada elemento tiene su propósito, su concepto, su mecanismo y su notación gráfica como lo muestra la figura 3.7, que son las caracteristicas de SADT.

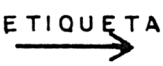
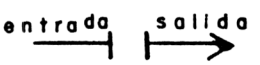
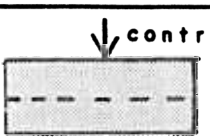
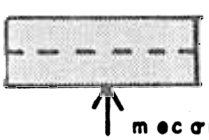
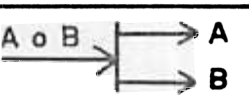
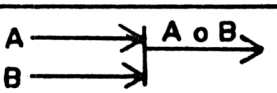
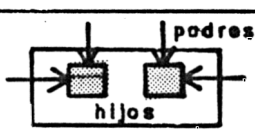
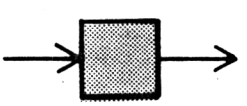
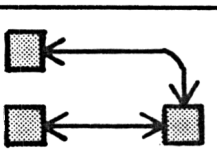
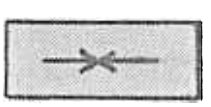
PROPOSITO	CONCEPTO	MECANISMOS	NOTACION	N
LIGADOR DE CONTEXTO	DENTRO FUERA	CAJA SA		A11
CONECTOR RELATIVO	DE/ A	FLECHA SA		A12
TRANSFOR-MACION	ENTRADA SALIDA	INTERFAZ SA		A13
CIRCUNSTANCIA.	CONTROL	INTERFAZ SA		A14
SIGNI CADO	SOPBRTE	MECANISMO SA		A15
MOSTRAR EXCLUSIONES	ALTERNA	RAMIFICACION OR		A222
		UNION OR		A223
INTERFAZ JERARQUIA DE DIAGRAMAS		FLECHAS FRONTERA SA		A231
MUESTRA COOPERACION	INTERCAMBIO DE RESPONSABILIDAD COM.	FLECHAS EN DOS SENTIDOS		A311
SUPRIME LOS DETALLES DE INTERCAMBIO	ADMITE 2 CON 1 FORMA EN LINEAS INF.	2-FORMAS 1-FORMA		A312
MUESTRA UNICA DESCOMPOSICION	REFERENCIA DE DETALLE	C-NUMERO DE PAGINA DEL DIAGRAMA		A23

FIG. 3.7 CARACTERISTICAS DE SADT

3.4.4 Modelo de Entidad-Relación.

En Ingeniería de "Software", uno de los principales problemas, es la composición de la base de datos dentro del diseño de los sistemas de procesamiento de datos. La integración de programas de aplicación a un conjunto de archivos requiere de módulos que conviertan datos y estructuras de acceso. Los modelos de base de datos permiten determinar esquemas tanto a nivel conceptual como a nivel físico con el objetivo de tener vistas independientes. El modelo de Entidad-Relación de Chen [CHEN76], es uno de los que actualmente tienen mayor difusión debido a que se ha estado incorporando a sistemas de CASE.

Las bases de datos son una solución al problema de integración de los sistemas; requiriendo, sin embargo, de una fundamentación de modelo para su concepción lógica conceptual y un sistema manejador para su funcionalidad. Para tal efecto, es necesario una metodología estructurada que permita sistematicamente convertir los requerimientos de los usuarios en un buen diseño de bases de datos. Siendo la aproximación de entidad-vinculo de Chen la metodología que a alcanzado una mayor popularidad dentro del modelado de datos y diseño de las bases de datos.

El modelo de entidad-relación trata con las entidades y las asociaciones con el propósito de tener una vista natural de la información más natural. En un primer nivel, contempla la vista de la información concerniente a entidades y relaciones, presentandose como objetos conceptuales en el mundo real; y un segundo nivel, el cual consideran los objetos conceptuales en sus representaciones directas de datos. El modelado de los datos emplea una poderosa herramienta gráfica denominada diagramas de Entidad-Relación.

En la construcción de los diagramas se incluyen rectangulos que significan el juego de entidades y rombos que significan el juego de relaciones. Las entidades están constituidas por atributos. Cuando el usuario define una entidad en un rectángulo el sistema solicitará un conjunto de atributos que definen a la entidad. Los atributos son dados de alta en un diccionario de datos para su uso y asociación posterior. El usuario define cual atributo es una llave primaria considerada como el atributo cuyos valores me permiten distinguir cada uno de los elementos del conjunto entidad. Las entidades son vinculadas con rombos que puede relacionar los elementos de una entidad de alguna de las siguientes formas: uno a muchos (1:n), muchos a muchos (n:m), y uno a uno (1:1).

Con el fin de ilustrar el modelo en la figura 3.8 damos un ejemplo de diagrama de entidad-relación para el diseño de una base de datos de una organización académica.

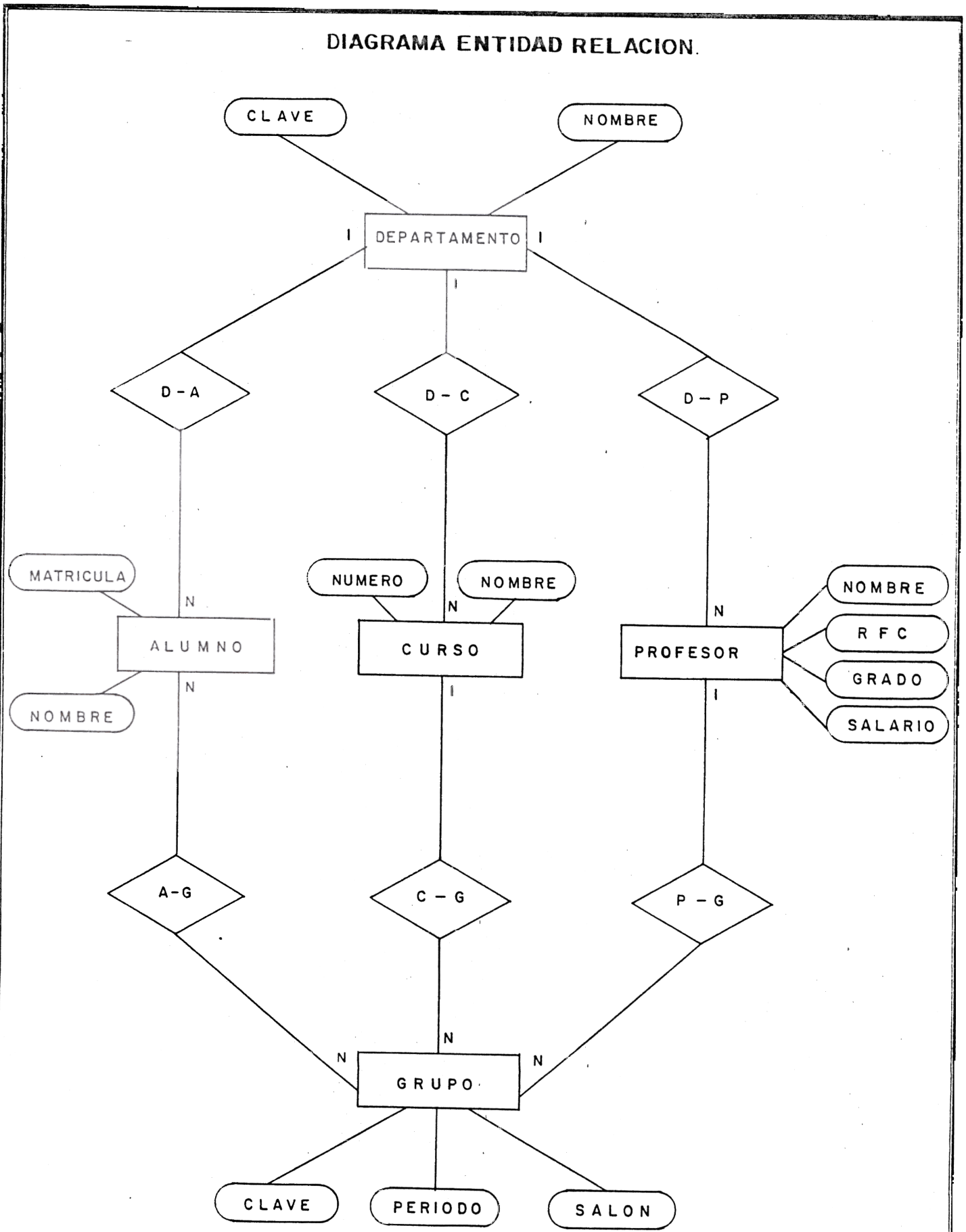


Fig. 3.8 Ejemplo de Diagrama de Entidad - Relación

3.5 LENGUAJES GRAFICOS PARA LA PROGRAMACION

Contrario al ambiente de programación que prevalece en lenguajes de texto como Pascal, Cobol, Lisp, etc.; los lenguajes iconicos -a veces llamados visuales- emplean fundamentalmente gráficas. Los símbolos gráficos son algunas veces usados como elementos decorativos, o algunas otras, integrados más definitivamente de acuerdo a sus reglas sintácticas y semánticas. Glinert en [GLIN84] precisa " para que un lenguaje o sistema de programación este diseñado en términos visuales, -opuesto a uno de texto- debe cumplir con las siguientes condiciones:

> Las entidades gráficas deberán ser diseñadas a un nivel alto de manera que estén disponibles para los usuarios como átomos. Los átomos podrán ser manejados en la programación o en el tiempo de ejecución. Las entidades gráficas de alto nivel son objetos geométricos (círculos y cuadrados, o cualquier otra imágen que represente "tokens") del alfabeto estandar de programación.
>
> 2 Los elementos gráficos pueden contener texto y números como componentes formando una parte integral del lenguaje y no solo como parte decorativa. "

Un lenguaje de programación es denominado icónico si el proceso de programación se basa esencialmente en seleccionar y/o arreglar apropiadamente iconos en una pantalla. Los iconos tienen significado y pueden ordenarse mediante reglas sintácticas, con la finalidad de representar un programa. Los sistemas icónicos datan de los sesentas con Sutherland [SUTH63], en donde introduce un sistema que usa gráficas interactivamente.

A partir de éste, se desarrollaron algunos otros trabajos -[NEWM68],[ELLI69] y [DENE74]-; pero, no es hasta mediados de la década de los setentas, cuando una mayor cantidad de sistemas de tipo gráfico empiezan ha ser implantados. En seguida de manera breve, se describen algunos sistemas enfatizando sus aspectos más relevantes.

3.5.1 Sistemas de Programación PYGMALION y PAD.

PYGMALION escrito en SmallTalk y desarrollado por la Xerox Palo Alto Research, es un sistema para la demostración y programación de tipo procedural [SMIT75]. El elemento primordial del sistema son los íconos, los cuales ofrecen varias contribuciones:

1) El ambiente de programación se basa en una comunicación hombre-máquina a través de pantallas gráficas.
2) El sistema proporciona al usuario un modelo interactivo.
3) El énfasis en la programación es de procedimiento.

La programación en PYGMALION es un proceso de diseño y edición en terminos iconicos. Los programadores interactuan con el sistema a nivel de pantalla con imagenes que ellos crean. Con la ayuda de un ratón el programador de PYGMALION demuestra las etapas de cómputo usando valores de datos muestra.

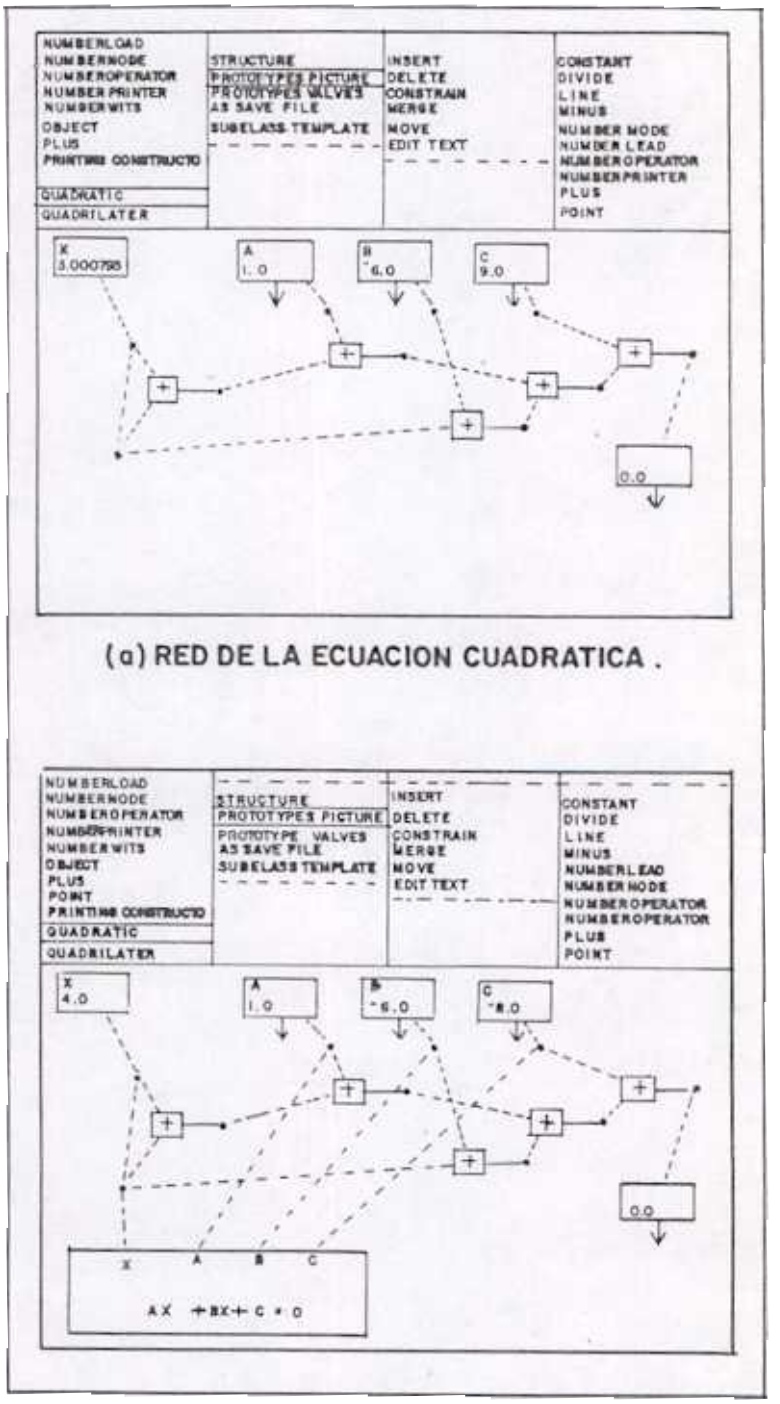
Una extensión al sistema PYGMALION es el Sistema Programming by Abstract Demostration (PAD) diseñado por Gael Curry [CURR78]. PAD igual que PYGMALION es un sistema para demostración y programación procedural; con la diferencia que PAD admite demostraciones que son llevadas sobre rangos de valores no acotados, llamados por Curry datos abstractos. El sistema fué implementado en XPL/G para una computadora Xerox Sigma/5 equipada con una terminal gráfica vectorial IMLAC PDS-1D y pluma de luz.

3.5.2 Lenguaje ThingLab.

ThingLab es otro sistema gráfico que tiene como objetivo representar gráficamente experimentos de laboratorio orientados a una simulación con restricciones [BORNI81]. El lenguaje Thinglab implementado en Smalltalk, incorpora objetos gráficos y simbólicos con un enfoque a dos diferentes clases de usuarios: 1) instructor de laboratorio, expertos en el campo y programadores en el uso de lenguajes de procedimiento de alto nivel; y 2) estudiantes. El instructor de laboratorio define la construcción de bloques para un dominio dado. Por ejemplo, simulación de circuitos eléctricos. Se dibujan iconos para representar gráficamente cada entidad como pueden ser resistores, baterias, alambres y medidores; especificando también las restricciones. A partir de esto, el estudiante emplea esos bloques para construir y explorar simulaciones particulares.

En la figura 3.9 (a) se muestra un ejemplo de Thinglab que trata con la solución de una ecuación cuadrática. En este caso se construye una dibujo de red para la solución de una ecuación cuadratica, en donde los coeficientes han sido integrados y la solución encontrada. El sistema de ThingLab puede decirnos si la fórmula esta guardada por medio de la definición de una clase de resolvedor cuadrático cuyas partes incluyen los cuatro números de nodos a, b, c y x; así como la restricción de la solución estandar. Es posible insertar una instancia del resolvedor cuadrático dentro de la red, mezclando sus números de nodos con los nodos apropiados existentes en la red (Figura 3.9 (b)).

FIG. 3.9 EJEMPLO EM THINGLSB PARA LA ECUACION CUADRATICA



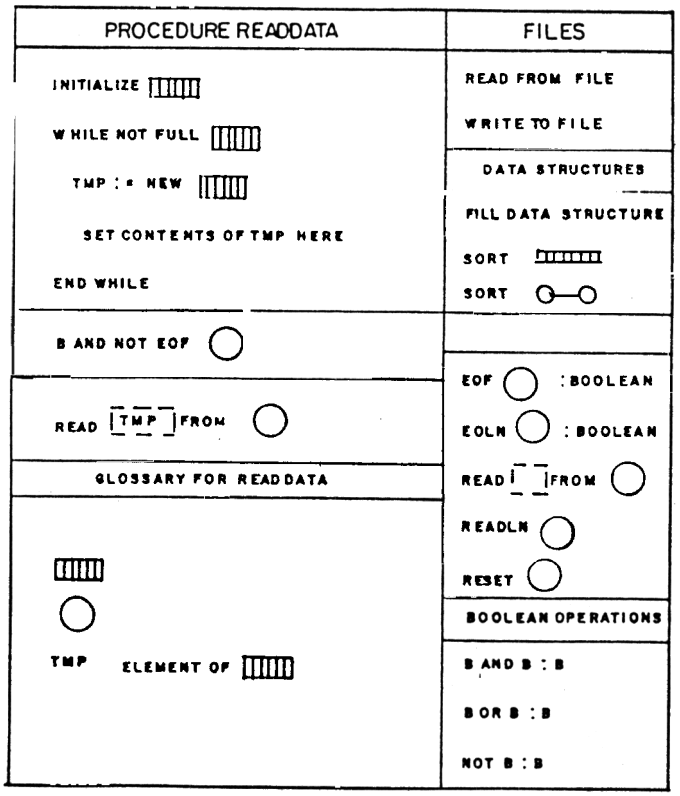
3.5.3 Sistema OMEGA.

El sistema OMEGA fué desarrollado en la Universidad de Berkeley [POWE83], con la finalidad de construir programas interactivamente con el señalamiento de estructuras gráficas y moviendose alrededor de la pantalla. El sistema tiene la posibilidad de desplegar varias ventanas a la vez para demostrar diferentes partes del programa, con la vista de entidades de la base de datos.

En Omega los iconos (pictografías) son abstracciones visuales que representan objetos almacenados con varias alternativas de representación. El usuario tiene la posibilidad de leer objetos en forma de sintaxis libre con el señalamiento de iconos. Sin embargo los detalles reales son especificados en forma de texto que se reensambla en un lenguaje de programación común.

Aunque Omega tiene un enfoque gráfico interactivo, mantiene un cierto predominio del enfoque convencional de programación de texto. No es muy claro en OMEGA, si la separación entre los objetos y su representación, realmente ayuda al usuario. El objetivo consiste en tener las representaciones de los objetos lo más concretos que sean posible para que puedan ser mostradas y manejadas facilmente por el usuario (Fig. 3.10).

FIG. 3.10 DESPLEGADO DE PROGRAMA EN OMEGA



3.5.4 Sistema PECAN

PECAN se diseñó y desarrolló en la Universidad de Brown en 1984 [REIS84], como un sistema que sirve para construir programas. El sistema opera en estaciones de trabajo Apollo requieriendo de pantallas de alta resolución para presentar información acerca de la sintáxis, semántica y ejecución del programa. Con el se presentan diferentes vistas del programa en una sola pantalla: diagrama de Nassi-Shneiderman, diagrama de flujo, lista de programa, árbol del "parser", tabla de símbolos, pila de ejecución y una muestra del dialogo de entrada-salida (ejemplo figura 3.11)

El objetivo de PECAN fué integrar completamente la programación y la graficación para el desarrollo de programas con un enfoque a lenguajes de programación de tipo algebráico. En el sistema PECAN el programador tiene la posibilidad de editar texto con el propósito fundamental de representaciones abstractas, combinando ambientes que ofrecen mejores características:

Retroalimentación inmediata de los errores de la sintaxis y semantica en que ocurre el usuario.

* Flexibilidad en la edición de texto por medio del uso inmediato de plantillas.

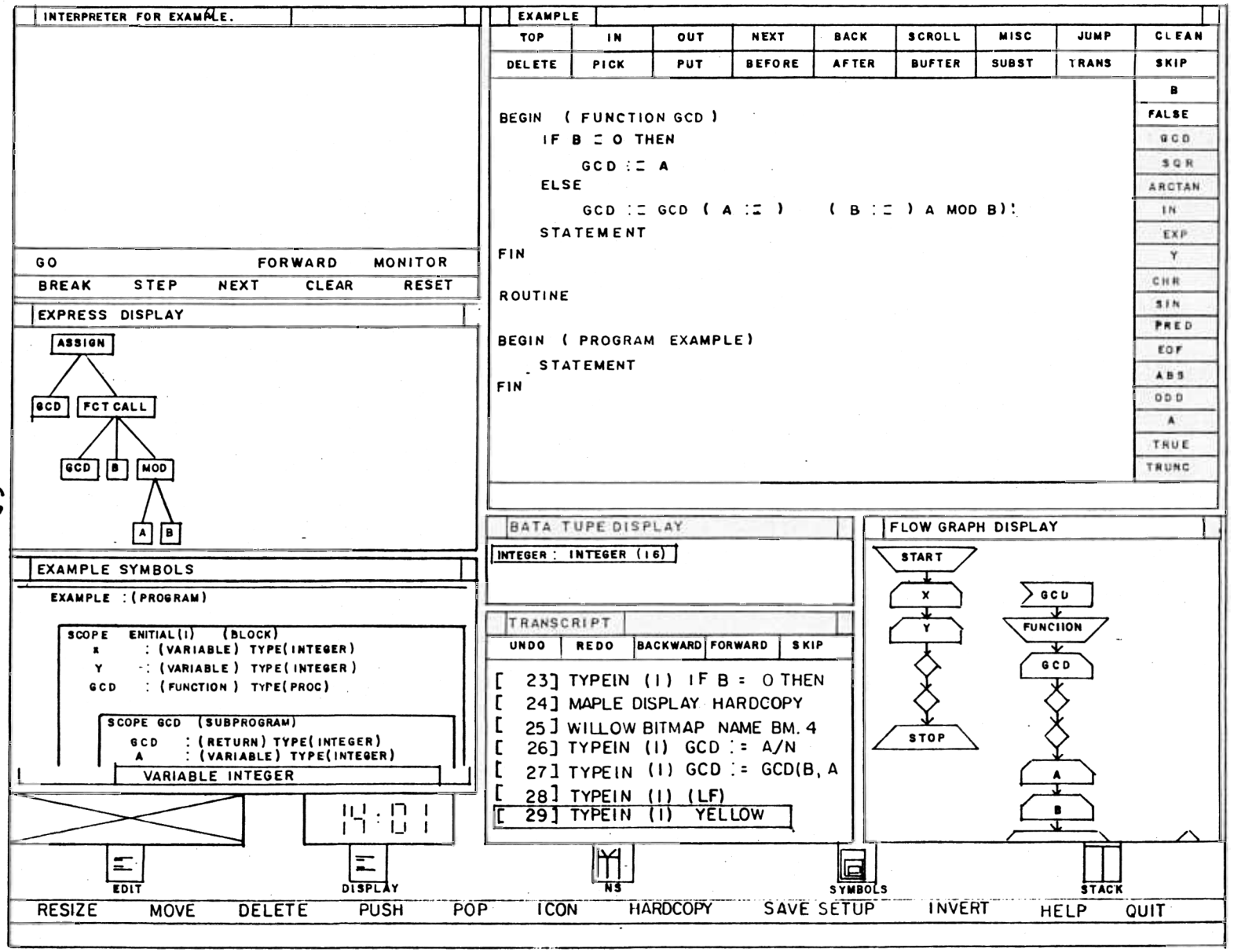
* El uso de menúes como alternativa a la edición de comandos.

* El desplegado de pantallas multiples con el fin de hacer uso de la pantalla eficientemente.

* Semántica incremental que admite que el programa sea compilado conforme es editado.

* Posibilidades de edición durante la ejecución.

FIG. 3.11 VISTA DE UN PROGRAMA EN PECAN.



5.5 Sistema Pict

Pict es el sistema "An Interactive Graphical Programming Environment" y fué creado por Ephraim P. Glinert y Steven L.Tanimoto [GLIN84], con la finalidad de ayudar a la implementación de programas, más bien como un algoritmo de diseño y selección. El sistema es un prototipo experimental capaz de construir programas pequeños pero no triviales.

El sistema Pict mantiene un ambiente gráfico de programación en donde los usuarios programan con dibujos que se editan en una pantalla de color como si fueran rompecabezas con una yuxtaposición de componentes predefinidos. Gráficas, color, sonido y animación son elementos que se usan para componer, editar y correr programas; con una sencilla estructura de comandos. El usuario se comunica con el sistema PICT por medio del señalamiento de iconos en un árbol de menú.

Varios prototipos de sistemas Pict han sido implementados en el compilador Pascal para la DEC VMS corriendo bajo Unix en una VAX 11/780. Los sistemas gráficos son pantallas Gould- De Anza de color de 512 X 512 o dispositivos raster con una entrada de color de 1024.

Trabajar con Pict/D es trabajar similarmente en el nivel de un lenguaje como Pascal y Basic, pero en forma visual. El menú principal del del sistema lo constituye un despliegue de cuatro iconos de arriba hacia abajo: Los de programación, representados por un pequeño diagrama de flujo. Borrado, con una representación de un borrador en un pizarrón. Editor de iconos, representado por una mano que sostiene una pluma lista para escribir en una hoja de papel. Y el uso de biblioteca, representada por un estante de libros.

En la Universidad de Washington el sistema Pict/D ha sido usado como un proyecto experimental de programación en los cursos de Programación Pascal e Introducción a Ciencias de la Computación. El objetivo fué tener una evaluación del sistema con una estadística de usuarios considerando aspectos de: facilidad de uso, facilidad de aprendizaje del lenguaje, rapidez de programación, entre otros. En el artículo de Glinert [GLIN84], se tiene más información al respecto.

3.5.6 Sistema PegaSys

PegaSys es el acróstico de "Programming Environment for the Graphical Analysis of SYStems". Creado por M. Moriconi y D. F. Hare en la internacional SRI [MORI85], es un sistema experimental cuyo propósito es el de facilitar la explicación y documentación de los programas diseñados.

En PegaSys un diseño de programa se describe por medio de una jerarquía de gráficas interrelacionadas, en la que cada una describe datos y las dependencias de control entre las entidades. Mientras las entidades son: subprogramas, procesos, módulos y objetos de datos entre otros; las dependencias son aquellas que se incluyen en cartas de flujo, diagramas de flujo de datos y lenguajes de interconección de módulos.

Tres peculiaridades principales son las que califican a PegaSys:

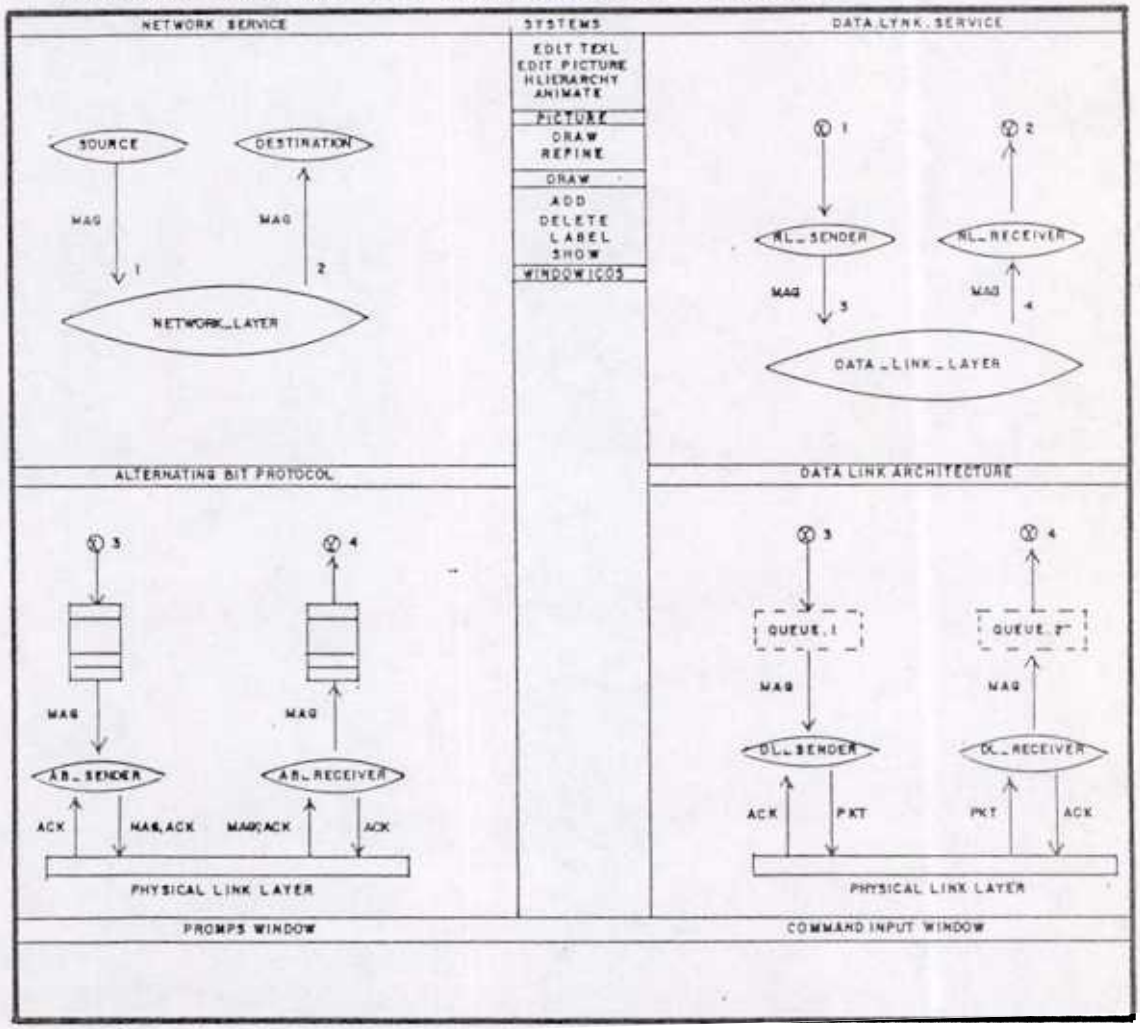
1) La verificación del significado sintáctico de las figuras.
2) Una guía en la aplicación de las reglas de diseño a través de la descomposición jerárquica de un diseño.
3) Determinación de si un programa ajusta a su documentación pictórica.

En PegaSys el significado de una figura se basa en el tratamiento gráfico y el de estructura lógica, en donde una figura es presentada como una estructura gráfica compuesta de iconos y sus propiedades, tales como: tamaño y localización. Los iconos en una figura corresponden a predicados en la representación lógica de una figura, cuya estructura denota un concepto computacional completo.

El sistema interacciona en terminos de iconos cuyas operaciones gráficas se tratan con restricciones lógicas que se imponen por el cálculo de formas. Con las restricciones aseguramos que las operaciones gráficas tengan sentido computacional, en donde el manejo gráfico depende del mapeo uno a uno del significado computacional de los iconos hacia los predicados. Las restricciones sintacticas y semánticas se aplican al manejo gráfico.

En la figura 3.12 mostramos un ejemplo de un primer nivel de jerarquía conteniendo varios iconos: cuatro elipses, un rectangulo, lineas de flecha y cadena de caracteres. La descomposición jerárquica de las figuras se efectúa de acuerdo a las reglas de diseño de PegaSys. Cada nivel en una jerarquía es una descripción de un programa a un nivel particular de detalle; formado por una secuencia de refinamientos que inmediatamente preceden al nivel de una jerarquía (Figura 3.12 (b)).

FIG. 3.12 DISEÑO DE PROGRAMA EN PEGASYS



.6 SISTEMAS ICONICOS PARA LA ADMINISTRACION Y EL PROCESAMIENTO DE DATOS.

Es un hecho que el diseño de lenguajes de muy alto nivel enfocados a la administración reviste un gran interés procurando desarrollos recientes en metodologías de diseño y programación. Dentro del ambiente de la administración el uso de lenguajes visuales y la graficación por computadora han mostrado ser bastante utiles para diferentes tareas como son: la documentación, el diseño, el análisis y el desarrollo de los sistemas administrativos.

Los diversos propósitos que se tienen por parte de los sistemas administrativos de: preparar, gestionar y procesar documentos; así como, el de ayudar a la toma de decisiones. Han han requerido de nuevas facilidades para que el usuario pueda interaccionar en diferentes contextos, permitiendo explotar de una manera mas eficiente los sistemas. Las estaciones de trabajo y los dispositivos gráficos es tecnología relevante que ha servido para crear interfaces interactivas con un sentido altamente visual. Bajo este ambiente nuevos sistemas de "software" presentan al usuario un estilo gráfico unificado, en donde los diversos propósitos se integran en una clasificación de sistemas para la automatización de las oficinas y procesamiento de datos:

a) Sistemas sustentados en modelos conceptuales concretos que representan la organización y permiten automatizar las gestiones de la administración. Con ellos el usuario puede interactuar gráficamente para crear, modificar y enviar documentos. Los administradores pueden modelar y simular sistemas dinámicos administrativos de gestión.

b) Sistemas de base de datos para que el usuario almacene, transforme y consulte la información. Procurando las metodologías enfocadas a la solución de problemas en dominios específicos que se tienen en Inteligencia Artificial es posible integrar sistemas de información inteligentes que ayuden en un gran extento a la toma de decisiones.

c Sistemas y lenguajes de programación para el desarrollo de aplicaciones dentro del dominio especifico del procesamiento de datos, que en general, se sirven de una extensa base de datos. Extensiones naturales a la programación de sistemas se tienen dentro del diseño y análisis de los mismos, todo esto incluido dentro de la tecnología CASE.

3.6.1 Sistema STAR.

El proyecto STAR de Xerox ha creado uno de los sistemas, que quizás, es el más representativo de los sistemas icónicos para la automatización de oficinas. El sistema esta constituido tanto por "hardware" compuesto de estaciones de trabajo para oficinas como del "software" que es una interfaz amigable para el usuario. El sistema integra en un ambiente gráfico un conjunto de iconos muy representativos de las funciones y objetos de datos que se encuentran en las oficinas. El sistema de interfaz STAR usuario-estación de trabajo tiene como objetivo presentar al usuario vistas orientadas a objetos que representan la funcionalidad de la oficina, e información que maneja el usuario en forma encapsulada por gráficos del sistema.

Por ejemplo, el nivel más alto de la carpeta se encuentra una colección de iconos que considera objetos globales de datos como son: los documentos, los archivos y los folders. Y como objetos funcionales: impresoras, cajas de correo, gabinetes de archivo, entre otros. En una jerarquía dentro de los objetos de nivel alto se tienen nuevos objetos como son: los caracteres, los parrafos y los objetos gráficos. Para que el usuario señale con un ratón pequeñas figuras que representan las funciones como suelen ser: abrir, cerrar, mover, copiar y borrar, entre otras.

El diseño del sistema STAR fué influenciado fuertemente por la interfaz de un ambiente de Smaltalk el cual tiene una orientación a objetos y su desarrollo se llevo al cabo con el lenguaje de programación Mesa, dando como consecuencia un producto el cual tiene una fuerte orientación a objetos. Lo relevante al desarrollo de STAR es la extensiva funcionalidad de la oficina y la interfaz gráfica amigable, que permite al usuario preparar, crear y gestionar documentos.

A pesar de los claros beneficios que se tienen con START como lenguaje visual para la automatización de las oficinas; para el desarrollo de programas de aplicación ha sido necesario esperar resultados en algún otro lenguaje. El lenguaje que ha sido acoplado es CUSP [PURV83], que es un lenguaje de programación el cual realiza una programación de texto con procedimientos, variables y parámetros.

3.6.2 Sistema OfficeTalk-Zero.

OfficeTalk-Zero es un prototipo de sistema de información para oficinas diseñado por W. Newman y Tim Mott; con una compleja implementación por parte del grupo de investigación en oficinas de "Xerox Palo Alto Research Center (PARC)" en multicomputadoras interconectadas en red con alta velocidad en comunicación [THAC79].

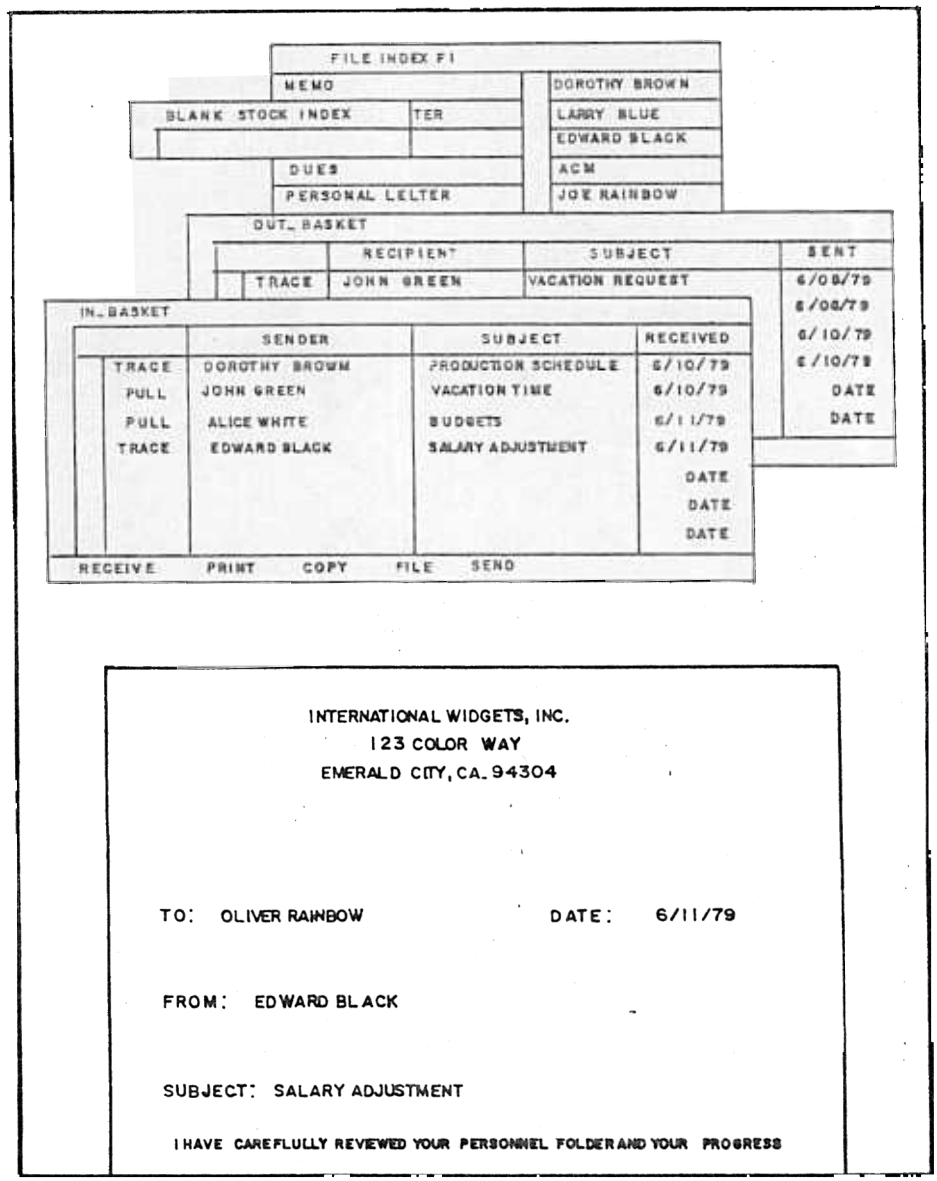
El sistema es un programa distribuido que trabaja minimamente con una minicomputadora con pantalla de rayos catódicos y otra comunicada en red que sirve como servidor de archivos. El servidor mantiene una base de datos que contiene todas las transacciones pendientes, tales como correo electrónico o un conjunto de formas en blanco para ser usadas en aplicaciones particulares.

Para implementar una aplicación particular de Officetalk un conjunto de formas en blanco deberán ser diseñadas y enteradas a la base de datos. El modelo se basa en representaciones gráficas de las formas de papel que por medio de ventanas el oficinista puede manejar haciendo uso del ratón. El diseñador de formas tiene la facilidad de alargar o ensanchar la forma y moverla con la ventana por medio del señalamiento del cursor en ciertas partes. Las ventana puede ser movida sobre toda la pantalla con la finalidad de observar la forma simple y aplicar algún comando del menú de officetalk que se encuentran visibles también en la representación de una forma (figura 3.13).

OfficeTalk integra un cierto número de ideas enfocadas a la interacción gráfica de manejo visual de formas [TEIT77], y conceptos de ventanas [HEWL78], aplicados también en la HP-3000. El "software" básico de graficación se basa en partes del "Core Systema" nivel 4 desarrollado por ACM SIGGRAPH, usando conceptos como recorte de ventanas y transformación de vistas [NEWM79].

De lo anterior se observa que el prototipo de Officetalk es un sistema dentro del contexto de la automatización de las oficinas, que integra una serie de facilidades en un ambiente visual con la finalidad de preparar, comunicar y gestionar documentos; teniendo además, facilidades para definir especificaciones de los procedimiento para las actividades de una oficina. Sin embargo, tiene fuertes limitaciones como un sistema de consulta en base de datos y el procesamiento de datos.

FIG. 3.13 IMAGEN DE FORMAS EN OFFICE TALK.



3.6.3 Sistema Quinault.

El sistema Quinault creado por Gary Nutt y Paul A. Ricci [NUTT81]; fué implementado en un sistema Xerox Alto, con pantalla refrescable de muy alta resolución y red de comunicación externa Ethernet. Tomando la ventaja de todo un ambiente de "hardware/software", Quinault fué escrito en el ambiente de programación Mesa que se encuentra implementado en el tope del sistema operativo Ejecutivo Alto, haciendo uso de su editor de texto, compilador, ligador y un espulgador simbólico. La filosofía basica del sistema es el de prescindir lo más posible del teclado, incluyendo en su lugar el ratón con entrada de selección en una salida gráfica.

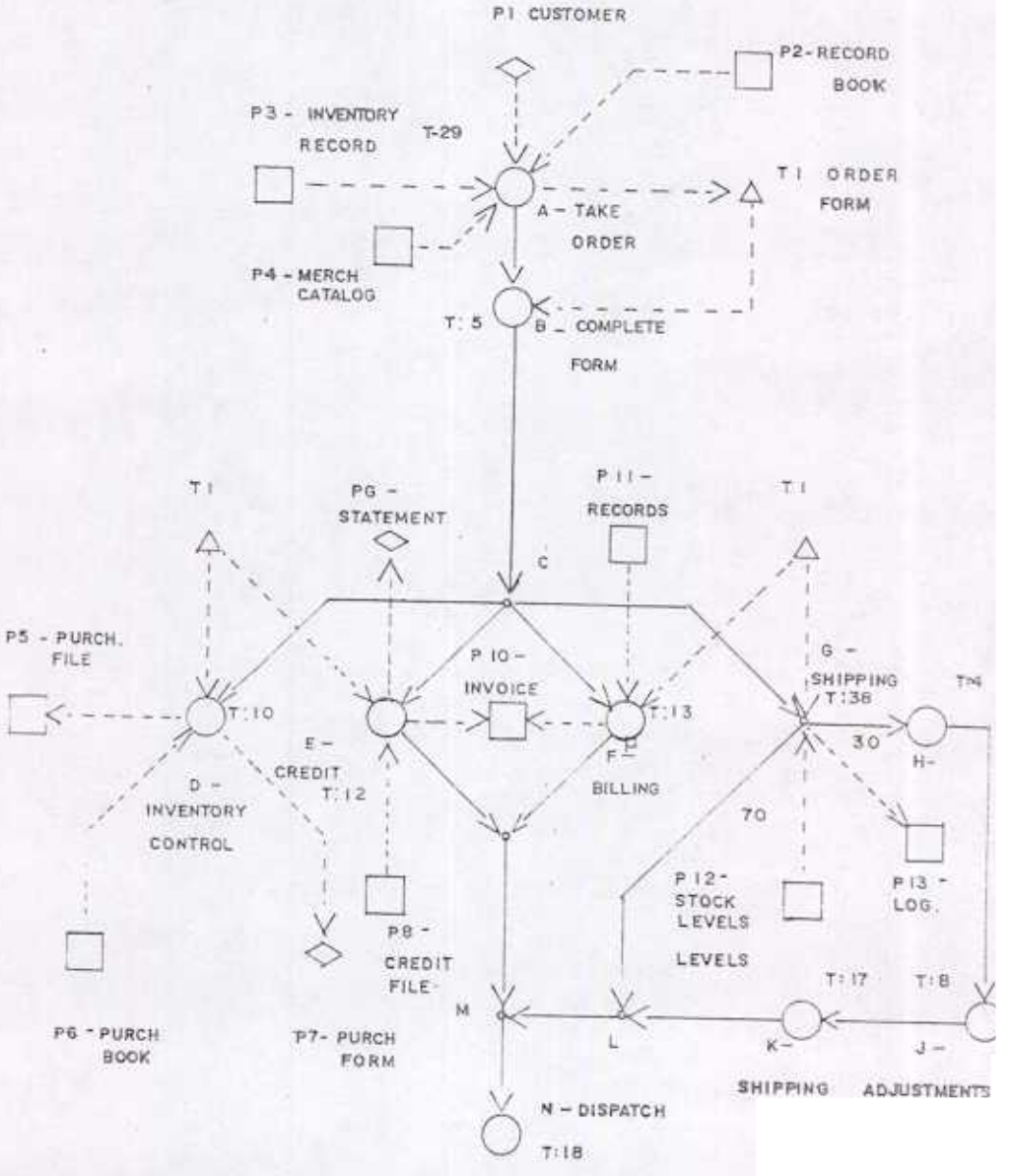
Quinault es un sistema automatizado que tiene por objetivo construir, analizar y simular modelos de oficinas. Por medio de una interacción gráfica permite entender las estructuras de las oficinas y el comportamiento de los sistemas de información, integrando el formalismo gráfico "Information Control Nets (ICN)" definidas por G. Nutt en [NUTT79].

ICN es un modelo gráfico de datos y control de flujo en oficinas que sirven para componer un conjunto de actividades. En el representamos las actividades por circulos y los medios de almacén de la información por polígonos. La referencia de los datos se representa por un arco entre un nodo almacén y un nodo actividad, pudiendo estar interconectados por otra clase de arcos que representan restricciones de precedencia en un conjunto de actividades. La semántica admite distinciones en sus representaciones esquemáticas de circulos grandes y pequeños; con sensibilidad de contexto claros y obscuros. Los circulos obscuros representan conjunciones lógicas (AND) y circulos claros representando disyunciones lógicas (OR).

El sistema consiste principalmente de un editor, un analizador y un simulador. El editor gráfico consiste de una interfaz que usa fundamentalmente el ratón para tomar de un menú de color rojo iconos del modelo ICN. Un menú de color amarillo contiene los comandos como: etiqueta nodo, etiqueta tiempos de actividades o pone probabilidades; que se deben anotar a la gráfica. El editor permite almacenar la ICN permanentemente para posteriormente poder analizar estáticamente la descripción y calcular la óptima a través de transformaciones.

El analizador proporciona al usuario que capacidades significativas no se encuentran disponibles en los sistemas manuales, tomando tiempos y probabilidades de las actividades. Finalmente, el simulador de Quinault usa la descripción de ICN para animar un flujo de transacción. En la figura 3.14 muestro un caso de procesamiento de ordenes desarrollado en Quinault.

FIG. 3.14 EJEMPLO DE PROCESAMIENTO DE ORDENES EN QUINAULT (MODELO ICH)



3.6.4 Query-By-Example y System Business Automation.

QUERY-BY-EXAMPLE (QBE) es el que encabeza la lista de los lenguajes de consulta de tipo iconico basados en el modelo de datos relacional. Creado por Zloof en 1974 [ZLOO74], ha sido usado ampliamente como un lenguaje de no-procedimiento y programación que tiene una sintáxis de dos dimensiones, debido a su naturaleza gráfica de poderse mover sobre una tabla. En este sistema las operaciones de consulta, actualización y definición en la base de datos se efectua con el llenado de un "ejemplo", que es un objeto gráfico que semeja formas o tablas (Figura 3.15 (a)).

En QBE las consultas se plantean mediante "ejemplos" usando variables de dominio y constantes; las cuales, llenan esqueletos de tablas para formar patrones de tuplas que están en las relaciones. QBE tiene facilidades como: uso de caja de condiciones que permite elaborar expresiones complejas empleando varios campos; especificación de vistas y creación de nuevas relaciones.

Desde 1978 que fué liberado el primer sistema de QBE por la IBM, QBE ha sido objeto de un amplio reconocimiento por parte de los usuarios como un lenguaje para: la definición, la actualización y recuperación en una base de datos relacional. Sin embargo, las limitaciones que tiene en el manejo de los datos mantiene restricciones para el desarrollo de programas de aplicación; situación que condujo a extensiones naturales de QBE a un nuevo lenguaje denominado SBA.

SBA "System for Business Automation" [ZLOO77] y [ZLOO81], es un lenguaje de programación con la misma filosofía de QBE que contiene tres componentes: Query-By- Example lenguaje para base de datos; lenguaje de programación como generador de aplicaciones; y el lenguaje de especificación de sistemas administrativos para describir las interrelaciones entre las aplicaciones (Figura 3.15 (b)). Un programa en SBA esta compuesto de una colección de transacciones en QBE sobre tablas o más objetos de datos complejos tales como formas y reportes. El programa usa ejemplos por especificación de invocación de programas, parámetros, disparos e interacción de usuarios.

Debido a la importancia que tiene QBE como representante de toda una clase de lenguajes de consulta visuales basados en la especificación en terminos de esqueletos de tablas y objetos de datos. En el capítulo seis doy una serie de ejemplos en donde se comparan la solución de los problemas en los lenguajes de QBE, SBA y el lenguaje LIDA; tanto como lenguaje de consulta como para el desarrollo de aplicaciones.

RELACION

A 1	A 2	A3	A 4	. . . .	An

CONDICIONES

FIG. 3.15 (a) RESUMEN DEL LENGUAJE Q B E

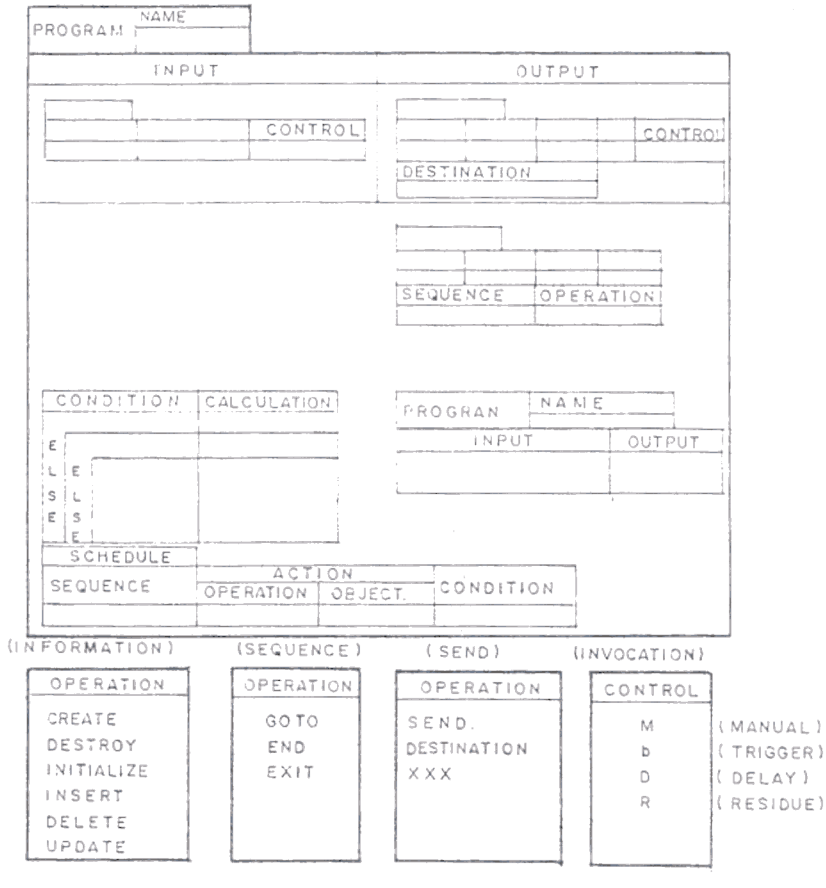


FIG. 3.15 (b) RESUMEN DEL LENGUAJE S B A

3.6.5 SDMS "Spatial Data Management System"

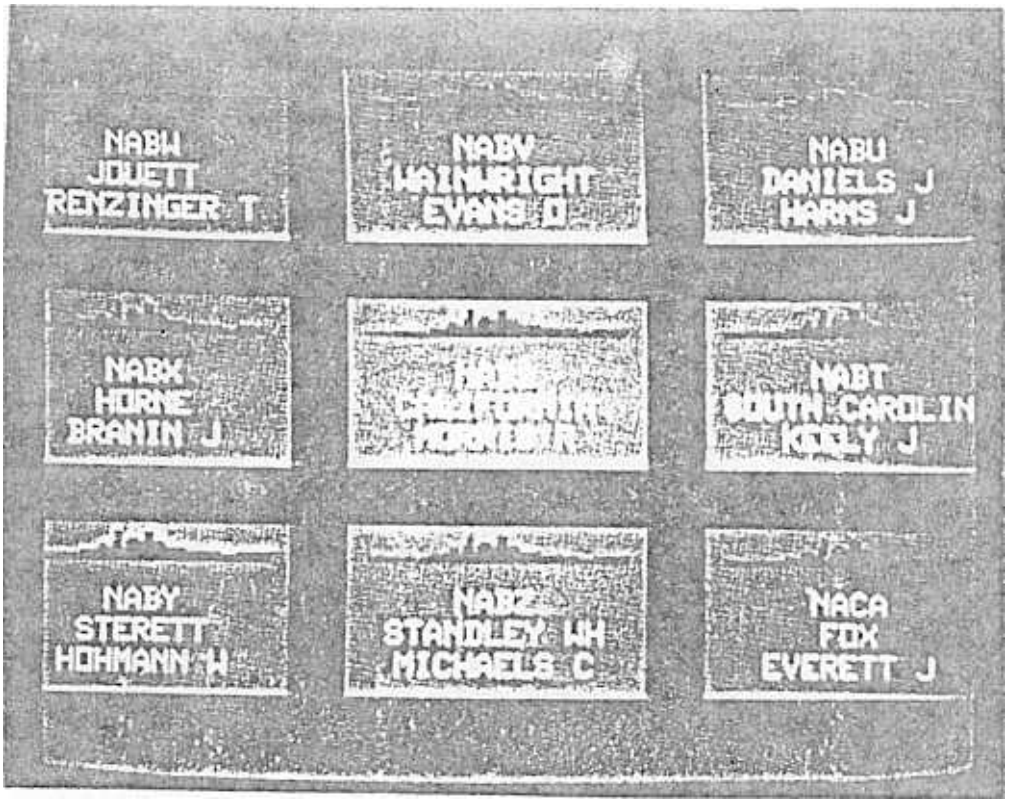
Christopher F. Herot creó "Spatial Data Mangement System" (SDMS) [HERO80], con la principal contribución del concepto de datos espaciales. El desarrollo a cargo de la compañia Computer Corporon of American estuvo soportado por un sistema complejo de gráficación en el que se incluían varias pantallas gráficas raster, discos digitales para almacénar imagenes como arreglos de bits y videodiscos ópticos.

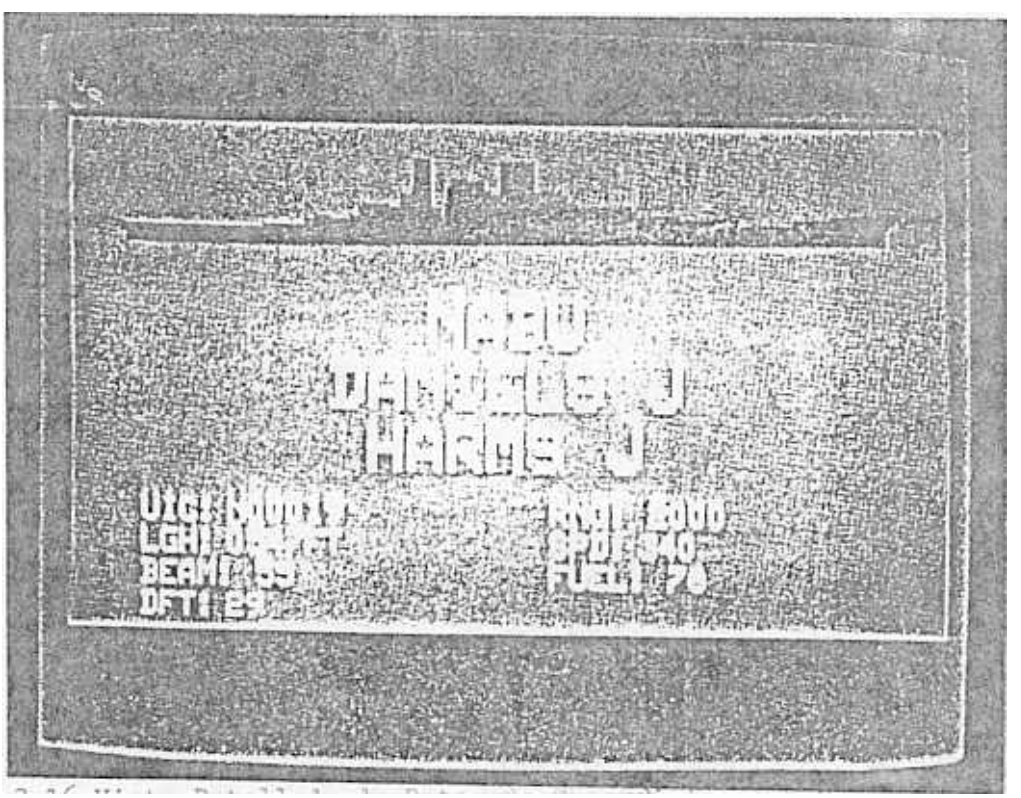
El uso del concepto de datos espaciales por parte del sistema SDMS permite explorar de forma simple la base de datos sin tener la necesidad de conocer el contenido y la organización de la misma base. Basandose en el modelo de datos relacional el sistema despliega el resultado de las consultas con iconos y textos, teniendo siempre una presentación gráfica directa de los datos.

Para el funcionamiento del sistema es necesario de dos pantallas "raster" que permitan visualizar la información mediante una colección de iconos en una vista global de la base de datos, y una local que restringe el campo visual pero aumenta el detalle de los mismos. El efecto de acercamiento efectuado por la aplicación un rectangulo controla su posición mediante el manejo de un "joystick". La información enmarcada en el rectangulo puede aumentar de tamaño haciendo girar el "joystick" en sentido de las manecillas del reloj. Simultaneamente al aumento de tamaño de los iconos la información se puede ver más detallada.

El sistema SDMS es un sistema icónico que presenta la información en términos de un grupo de iconos que tienen un dibujo representativo asociado a su contenido. El usuario con esta representación gráfica acerca de la información puede formular consultas simples navegando en la base de datos con poco conocimiento de ella. Sin embargo, en SDMS no es posible realizar consultas complejas.

Debido a que SDMS, representa una clase muy particular de lenguaje visual de consulta, en donde los resultados de las consultas se despliegan mediante la presentación de iconos y texto, muestro un ejemplo de base de datos de barcos transferido del articulo de Christopher [HERO80]. En el se muestra una colección de iconos de una base de datos de barcos, en donde cada uno de ellos, es el dibujo de un barco con datos asociados al mismo. En un acercamiento a la base de datos un icono es enfocado para observar su información detallada (figura 3.16).


NABW
JOUETT
RENZINGER T
NABV
WAINWRIGHT
EVANS D
NABU
DANIELS J
HARMS J
NABX
HORNE
BRANIN J
NABT
KEELY J
NABY
STERETT
HOHMANN W
NABZ
STANDLEY WH
MICHAELS C
NACA
FOX
EVERETT J



3.6.6 Sistema TIMBER.

TIMBER creado por Michael Stonebraker y Joseph Kalash en 1982, fué desarrollado en la Universidad de California en Berkeley [STONE82]. El sistema consiste fundamentalmente de una interfaz gráfica que facilita la navegación en una base de datos soportada por el sistema INGRES basado en un modelo de datos relacional. El hecho de que TIMBER este soportado por INGRES le ha dado la posibilidad de manejar datos geográficos

El sistema gráfico TIMBER -a diferencia de los que usan vistas en esqueletos de tablas como QBE- los objetos de la base de datos son representados por iconos, teniendo la posibilidad de almacenarlos de la misma forma que los textos y las relaciones de formato fijo, que también son tratados como objetos.

El usuario interacciona con el sistema por medio del cursor que se posiciona en alguna ventana, y al través de comandos que solo tienen su efecto en dicha ventana se ligan relaciones que resultan de: un comando de consulta; una vista sobre la base de datos; y una relación ya existente. Una vez que se tiene ligada la relación a la ventana, podemos editar comandos en forma de texto para: realizar una proyección; buscar el primer icono o renglón que satisfaga a una expresión booleana; seleccionar todas las tuplas en la ventana actual que satisfaga una condición; y borrar los iconos o renglones que no satisfagan una cierta expresión booleana.

.6.7 Sistema "Living in a Database"

Living in a Database es LID y es un excelente representante de los lenguajes gráficos basados en el modelo de datos de Entidad-Relación. A diferencia, de los anteriores (QBE, SDMS y TIMBER) que se basan en el modelo de relación. En 1984 en el departamento de Inteligencia Artificial de MIT Dennis Foog creó LID [FOOG84], con la idea principal de tener un sistema cuya interfaz permitiera que en todo momento presentára una pantalla únicamente con las entidades y las relaciones asociadas a una cierta tupla.

LID resuelve consulta a una base de datos atravès de diagramas de Entidad-Relación. Los diagramas constan de rectángulos que contienen entidades y se encuentran relacionadas entre si por arcos que las conecctan (Fig. 3.17). En LID cuando se requiere saber acerca de la existencia de un elemento entidad relacionado a su participación en otra entidad se sigue una consulta en dos pasos.

Suponga que tiene una base de datos con una representación de diagrama entidad-relación según la fig. 3.17. Si queremos saber cuales son los grupos de trabajo en los cuales trabaja el empleado "Juán López". Entonces seguimos los dos pasos siguientes:

a) Primero encontramos la tupla "Juán López" dentro de la entidad EMPLEADOS.

b) Posteriormente con el vinculo que existe con la entidad de GRUPOS se encuentran todos los elemntos de entidad que se están relacionadas con la anterior.

Las entidades y relaciones son las que están asociadas con la propuesta por la consulta. Sin embargo, en el esquema general (Fig. 3.17) otros conjuntos de entidades están relacinados, pero dada la pregunta y la relación las otras tuplas no aparecen; razón del nombre del sistema: el usuario "vive" dentro de una tupla de la base de datos y puede examinar sus alrededores.

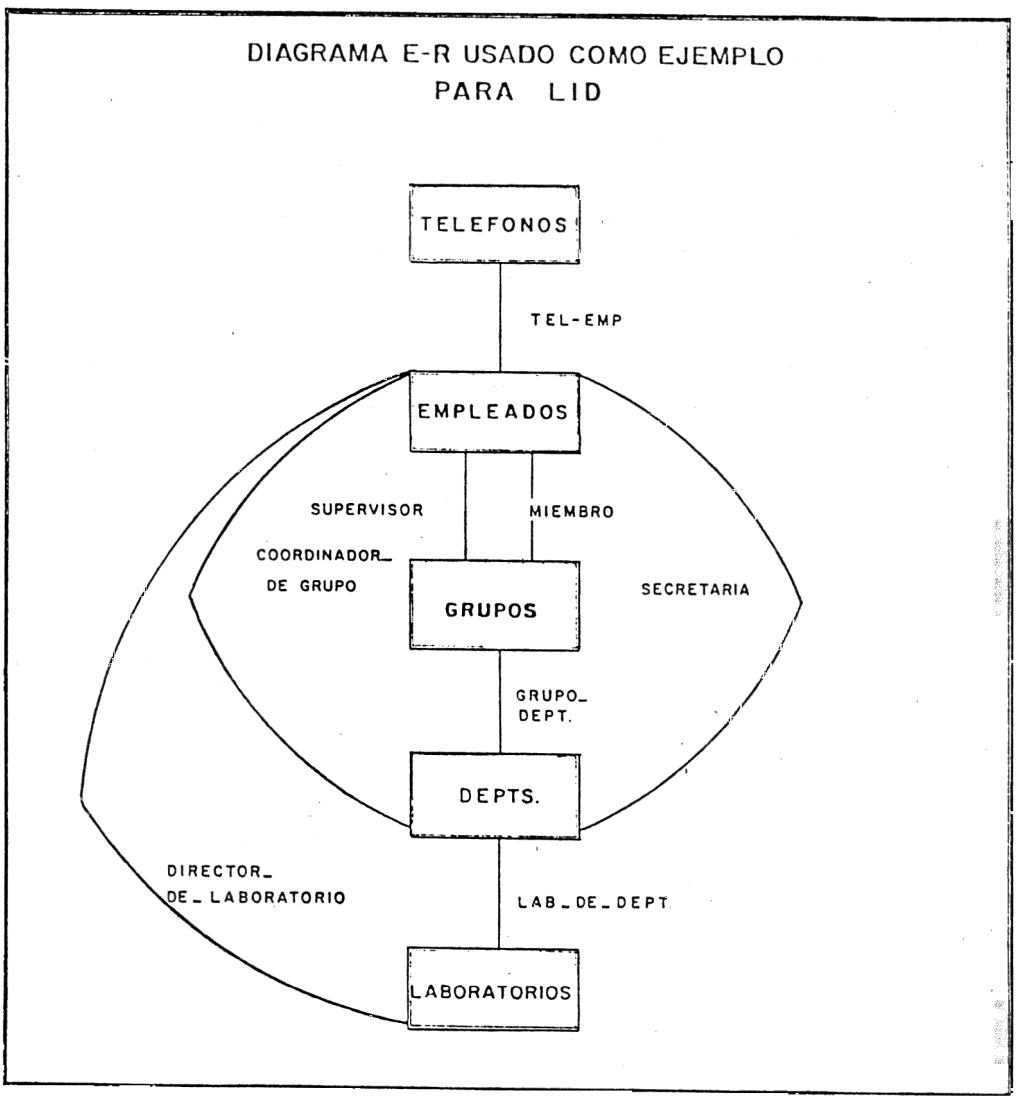


Fig. 3.17 Diagrama para Despliegue de Vistas en LID

3.6.8 iste FORMAL

FORMAL es el acróstico de "Forms-ORiented Manipulation Language" y es un lenguaje de programación orientado a formas. Realizado en 1985 por la IBM en el Centro Científico de los Angeles, es uno de los más recientes sistemas para desarrollo de aplicaciones que tiene la capacidad de computarizar de una manera simple un amplio rango de actividades de procesamiento de datos. El proyecto encabezado por Nan C. Shu [SHU 85], tuvo como finalidad desarrollar un lenguaje de muy alto nivel con la base del modelo de datos de formas, lenguaje orientado a formas y la aproximación usual del llenado de formas.

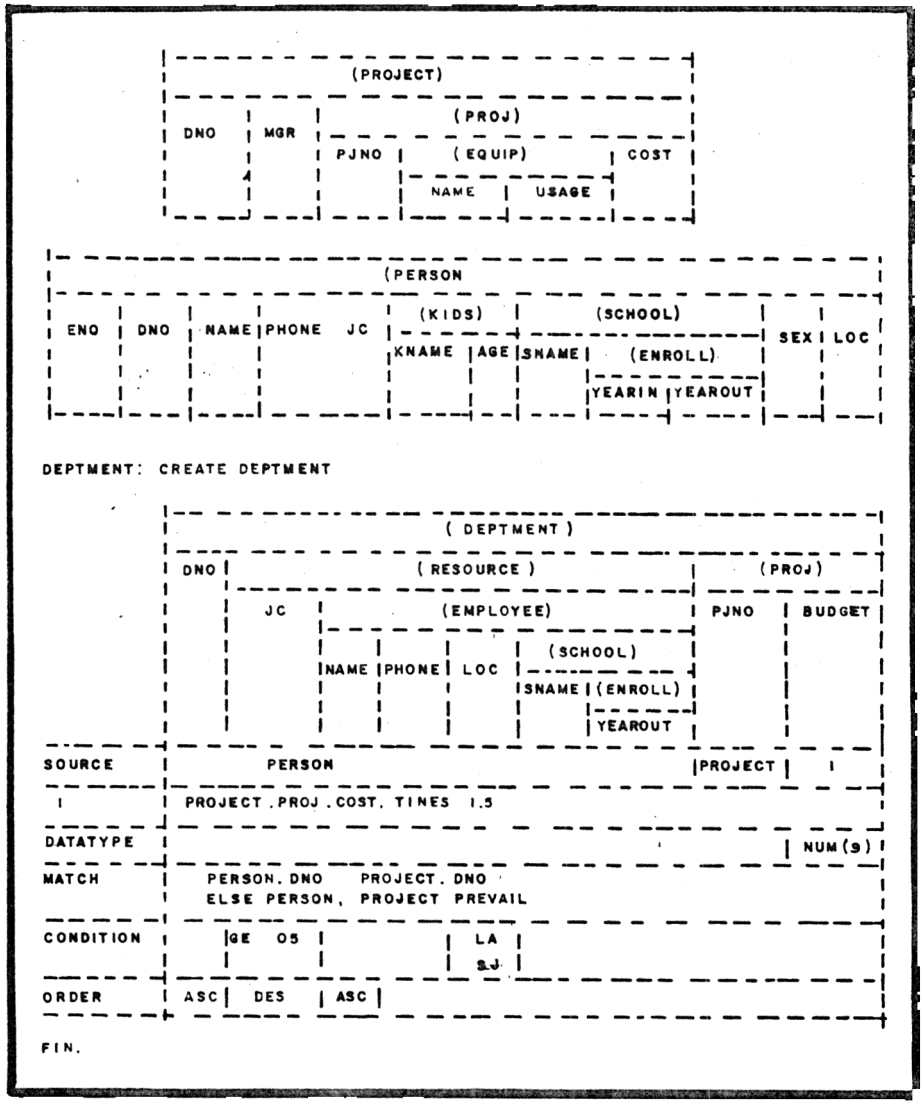
El modelo de datos de formas fué primero sugerido por Nan C. Shu en [SHU75], como una representación en dos dimensiones de datos jerárquicos, con la idea de proporcionar al usuario una ayuda visual para entender las operaciones de las formas de alto nivel. Un poco después Shu refina el modelo de datos de formas [SHU 82], ofreciendo una representacióon estructural que corresponde a una abstracción de formas de papel convencionales; dando origen al concepto fundamental de estructura de forma. Con este concepto es posible llegar al lenguaje de manejo de datos en donde las operaciones explicitas de alto nivel se transforman verdaderamente en especificaciones de no-procedimiento.

En este contexto no solo Shu ha mostrado interés en el uso de formas como representación visual de los datos. También, D. Luo y S.B. Yao adoptan el modelo de formas para el desarrollo de una interfaz de operación de formas por ejemplos (FOBE). El modelo de de tabla anidada y las operaciones algebráicas de alto nivel para el manejo de dichas tablas presentadas en [KITW84], son semejantes al modelo de forma-datos y las operaciones de formas (o Convert) informados por Shu [SHU 75] y Housel [HOUS76].

El sistema FORMAL para el desarrollo de aplicaciones con una apariencia visual y orientado a formas consiste de tres componentes principales: modelo de datos de formas, lenguaje orientado a formas y la usual aproximación de llenado de formas. La primer componente consiste en el modelo conceptual de datos que toma a las formas como objetos que tienen una colección de instancias con la misma estructura de datos. Los componentes de una forma pueden ser combinaciones de campos y grupos; donde un campo es la unidad más pequeña de los datos y un grupo es una secuencia de uno o más campos con la posibilidad de subordinar grupos. La segunda componente es el lenguaje de programación orientado a formas, llamado FORMAL.

Lo fundamental en este lenguaje consiste en llevar las actividades de procesamiento de datos en términos de un procesamiento de formas. En general, cada proceso de formas toma una o dos formas como entrada y produce otra forma de salida. El elemento final del sistema consiste en una comunicación entre el sistema y el usuario determinada con el llenado de una forma (ejemplo fig. 3.18).

FIG. 3.18 EJEMPLO EN FORMAL



3.7 RESUMEN Y TABLA COMPARATIVA DE SISTEMAS ICONICOS.

Dado el panorama de sistemas icónicos ya presentado en esta sección se presenta un resúmen de las principales características de algunos sistemas que pudiéran ser comparados con LIDA. Enseguida doy una lista de lenguajes con un resúmen de tres tablas que muestran las características y facilidades de algunos lenguajes de consulta a base de datos

TABLA DE ACROSTICOS Y NOMBRES

ACROSTICO	NOMBRE
ESCHER	
ER/FC	: Entity-Relationship/Formas de Captura.
GUIDE	: Graphical User Interface for Database Exploration.
gql/ER	: graphical query language for Entity-Relationship Database.
HIQUEL	: An Interactive Query Language to Define and use Hirarchies.
IQLEDB	: Interactive Query Language for External Data Base.
LID	: Living in a Database.
LIDA	: Lenguaje Iconográfico para el Desarrollo de Aplicaciones.
QBE	: Query-By_Example.
SDMS	: Spatial Data Management System.
TIMBER	

TABLA DE: MODELOS Y LENGUAJES.

MODELOS DE DATOS	LENGUAJES
RELACIONAL	TIMBER, IQLEDB, SDMS, LIDA QBE.
ENTIDAD-VINCULO	GUIDE, LID, gql/ER, ER/FC IQLEDB.
JERARQUICO	HINQUEL

TABLA COMPARATIVA DE LENGUAJES DE CONSULTA VISUALES

LENGUAJES DE CONSULTA

	FORMA GRAFICA	Q B E	SOMS	GUIDE	GQI/ER	HINQUE	ER/FC	ESHER	L I D	L I D A
CONSULTAS	ESQUELETOS DE TABLAS	▨				▨		▨		
	DIAGRAMAS EN LIDA VINCULO			▨	▨		▨		▨	
	FLUJOGRAMAS									▨
	REPRESENTACION DIRECTA DATOS		▨							
DESPLIEGE DE CONSULTAS	ESQUELETOS DE TABLAS	▨				▨				▨
	ICONOS TEXTO		▨							
	FORMA TABULAR			▨	▨			▨		
	FORMAS CAPTURA						▨			
	DIAGRAMAS BASE DATOS								▨	

En el capitulo 2 se trataron los sistemas MODEL II y BDL, mientras qué, en el capitulo 3 se consideran los sistemas visuales SBA y FORMAL; con la finalidad de tener una comparación con LIDA y ciertas conclusiones tenemos.

MODEL II es un LMAN que acepta las especificaciones del usuario en terminos de la descripción de datos y sus relaciones funcionales en forma de texto. Contrario a LIDA cuya presentación funcional es icónica estructurada en un flujograma, en donde los datos son objetos que describen el flujo de información.

El sistema MODEL II genera programas en PL/1 o COBOL, usando un modelo jerárquico para la base de datos, asociado fundamentalmente al tratado en COBOL. Aunque, actualmente LIDA sólo genera programas en código de cadena intermedio, es posible dejar un fuente en otros lenguajes como puede ser: SQL para base de datos o un lenguaje de programación como Pascal.

Debido a que MODEL II tiene el planteamiento de las especificaciones en forma de texto, el sistema contiene refinamientos con modulos de verificación de: la completitud, ambigüedad, e inconsistencia de las especificaciones. Aspectos que no tinene el sistema LIDA, debido a que los simbolos gráficos dan una determinación directa del significado, la elaboración del programa es más imperativa (ver en el capitulo 7 método de ejecución de un flujograma).

Por otro lado, BDL es un sistema que representa toda una línea de investigación de la IBM que prentende cubrir tres aspectos: la definición de formas con el sistema FDC, el flujo de documentos con el sistema DFC y la trasformación de documentos con el sistema DTC.

En principio podemos mencionar que BDL ofrece también facilidades gráficas con respecto a imagenes que definen formas cuyo enfoque es similar a Officetalk. Con la diferencia que en Officetalk el usuario no puede desarrollar programas y en BDL si es posible construir programas simples a partir de plantillas. Sin embargo, en el sistema BDL el lenguaje DTC de desarrollo de aplicaciones es no gráfico (ver figura 2.3).

Comparando con LIDA puedo mencionar que la principal contribución es la de tener un lenguaje para el desarrollo de aplicaciones eminentemente gráfico. De alguna forma es posible comparar la fig. 2.3 que procesa ordenes en BDL con el flujograma de la fig. 6.18 de facturación. Ambos, toman un jerarquización de anidamiento de cálculo desde ordenes hasta productos, considerando descuentos sobre clases. LIDA usa un modelo de datos relacional contrario al sistema de archivos que usa BDL.

Por otro lado, se tiene otra clase de sistemas, que de naturaleza gráfica, mantienen un enfoque visual directo hacia las tablas y formas (SBA y Formal). La semilla es QBE cuya representación gráfica son esqueletos de tablas con el manejo de objetos de datos.

SBA mantiene la caracteristicas de QBE unificando el modelo de bases de datos y las operaciones de programación. El sistema es no-procedural programable "por ejemplos", y hace uso de una base relacional con facilidades de la junta.

FORMAL es también un LMAN que extiende el manejo de los objetos al de Forma, para automatizar un rango ligeramente mayor de tareas comunes del procesamiento de datos. En el lenguaje el objeto de expresión visual principal son las formas con encabezados anidados; a diferencia de QBE que trabaja con tablas en un plano. En FORMAL manejamos esqueletos con encabezados que permiten ser acomodados en diferentes niveles para anidar o manejar en paralelo tablas asociadas. Dando como consecuencia la contribución de un tratamiento jerárquico con extensiones a los procesos de manejo de datos:

a) Alcances a uno o dos niveles del árbol jerárquico

b) Uso de "proyeccción", "restricción" y "junta" en esta estructura.

c) Derivación de nuevos datos por operaciones aritméticas.

d) Ordenamiento de instancias de formas dentro de otras formas.

Mientras FORMAL y SBA se basan principalmente en el concepto de formas administrativas y su concepto visual de esqueletos en donde el objeto visual es el dato y/u operadores que llenan los esqueletos. LIDA usa la representación de diagrama con un modelo de flujo de datos (pueden verse ejemplos comparativos en el capitulo 6). LIDA usa el modelo relacional de datos, sin jerarquías a ningún nivel, como se tiene en FORMAL.

A P I T J L O

MODELOS DE DATOS EN EL LENGUAJ LIDA

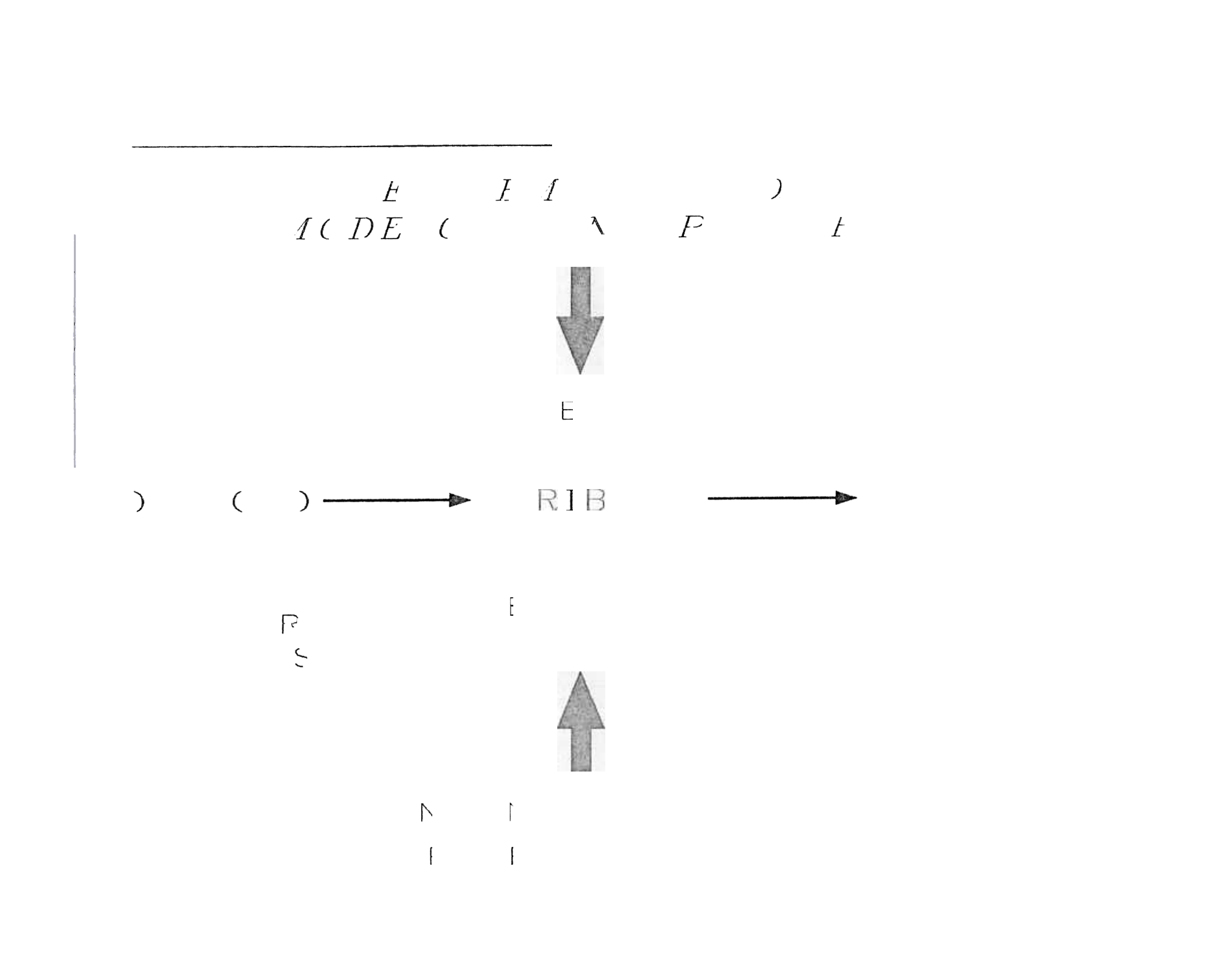
El objetivo de este capitulo es el de presentar el modelo de datos sobre el cual se basa LIDA. La orientación al procesamiento de datos del sistema considera que una de las necesidades del lenguaje es usar una base de datos. Se requiere de la definición de un modelo de datos en la vista conceptual asociada a su estructura de organización física. En este capítulo expongo cómo el sistema considera al modelo de datos relacional en el nivel de vista conceptual, vinculandose a un nivel más bajo de descripción física determinado por el esquema de Descriptor de Archivos.

En el nivel conceptual se definen un conjunto de objetos genéricos y con la aplicación de operadores se procura una representación directa en el dominio de discurso. Por ejemplo, en el modelo relacional de Codd [CODD70], se establecen objetos denominados relaciones sobre los cuales se proponen un conjunto de operadores para transformar relaciones a relaciones. Y en el caso del modelo de entidades, los objetos de entidad se operan con los operadores de la teoria de conjuntos [SENK73].

Por el otro lado, en el nivel más bajo se debe establecer un esquema de descripción física que incluye objetos que no necesariamente son análogos a los que se manejan en el mundo real como son: archivos, estructuras de datos, registros, indices, apuntadores, etc. En este contexto tenemos asociados métodos, algoritmos y procedimientos; que sirven como mecanismos para resolver los problemas relativos al acceso y recuperación.

Tanto los sistemas manejadores de base de datos como los dedicados al desarrollo de aplicaciones poseen de un lenguaje de definición de datos. El Lenguaje de Definición de Datos (LDD) permiten definir los esquemas y subesquemas de las bases de datos, que en este sistema cómo ya se mencionó en el capitulo 1 lo representa ell módulo de descriptor. La contribución del sistema LIDA acerca del LDD es tener un lenguaje de tipo visual en donde trata principalmente el objeto de dato en esqueletos; considerando que la interfase despliega ventanas en la pantalla con un esquema de tablas. El tratamiento es completamente similar tanto para la definición de datos como para la captura.

En el sistema el modelo conceptual se asocia al esquema físico por medio de un mapeo sencillo de (objeto, atributo, valor) como puede observarse en la figura 4.1 de Abstracción de Datos. Es un hecho que algunos sistemas comerciales como son: Peterlee Relational Test Vehicle (PTR), Progress, PTR y ZIM toman mapeos simples para su representación y su estructuración.



4.1 EL MODELO RELACI NAL D DATOS.

El modelo relacional de datos fué propuesto por Codd en [CODD70], con la definición de: los objetos de datos de relación, una álgebra relacional y un conjunto de reglas de restricción, que normalizan la base de datos.

El objetivo es definir objetos de datos que representen entidades del mundo real de una forma uniforme. Todas las relaciones son vistas como tablas y cada renglón de la tabla tiene el mismo formato.

Se define una álgebra sobre el conjunto de relaciones, definiendo operadores que tienen como argumento relaciones, para dar cómo resultado relaciones. Se consideran los operadores binarios usuales booleanos de unión, diferencia e intersección (ver capitulo 5), añadiendo la selección, proyección y junta.

Formas normales son restricciones en base de datos para evitar ciertas estados de las base no deseables. Formas normales representan el grado de redundancia que se tiene con los datos de la base. Codd define tres formas normales:

Primera Forma Normal.- Un esquema de relación R está en primera forma normal (1NF), si los valores en el dominio no on listas o conjuntos de valores.

Segunda Forma Normal.- Un esquema de relación R está en Segunda Forma Normal (2NF), si está en (1NF) y cualquier atributo no primo es completamente dependiente de cualquier llave de R.

Dada una relación R, un atributo A en R y un conjunto de dependencias funcionales F sobre R. Decimos que un atributo A en R es primo con respecto a F, si A está contenido en alguna llave de R. De otra forma A no es primo. Una llave en una relación R es por definición una combinación de atributos con la propiedad de identificación única para tuplas y no redundancia.

Dependencia Transitiva.- Dado un esquema de relación R y un subconjunto X de R, un atributo A en R, y un conjunto de dependencia funcionales F. Decimos que A es una dependencia transitiva sobre X en R, si existe un subconjunto Y de R con X --> Y, Y -/--> X y Y -/---> A bajo F de tal forma que A no esta en XY.

Tercera Forma Normal.- Un esquema de relación está en Tercera Forma Normal (3FN) con respecto a un conjunto de dependencia funcionales F si está en 1NF y ningún atributo no primo en R es dependencia transitiva sobre una llave de R.

Codd menciona [CODD70], que una base de datos es relacional en un 75% si tiene definido el esquema de relación y el álgebra relacional. Si ádemas, esta normalizada se dice que es completamente relacional.

Un esquema de relación R se denomina con un nombre y se define como un conjunto finito de nombres de atributos

A1, A2,...., An

a cada nombre de atributo Ai se le asocia un conjunto Di correspondiente al dominio de Ai, descrito como Dom (Ai). Se considera la definición formal de una relación R enaria, como el subconjunto del producto cartesiano de n-conjuntos arbitrarios no-vacios, finitos o infinitos numerables.

Por ejemplo:

EMPLEADO (Nombre, Clave, RFC, Sueldo, Categoria

VUELOS Numero, Aerolínea, De, A, Llegadas, Salidas)

CLASE (Curso, Semestre, Profesor, Salón

Definición.- Dada una serie de conjuntos D1,D2, ..., Dn; nonecesariamente distintos el producto cartesiano de estos n conjuntos (denotado D1 x D2 x . . . x Dn), es el conjunto de todas las tuplas ordenadas (d1,d2, ..., dn) en donde cada di; i=1,2, ... n es un elemento de su dominio Di respectivamente. Se define una Relación R como un subconjunto del producto cartesiano de los Di.

Sea R el esquema de relación denominado como RESERVACION y definido como:

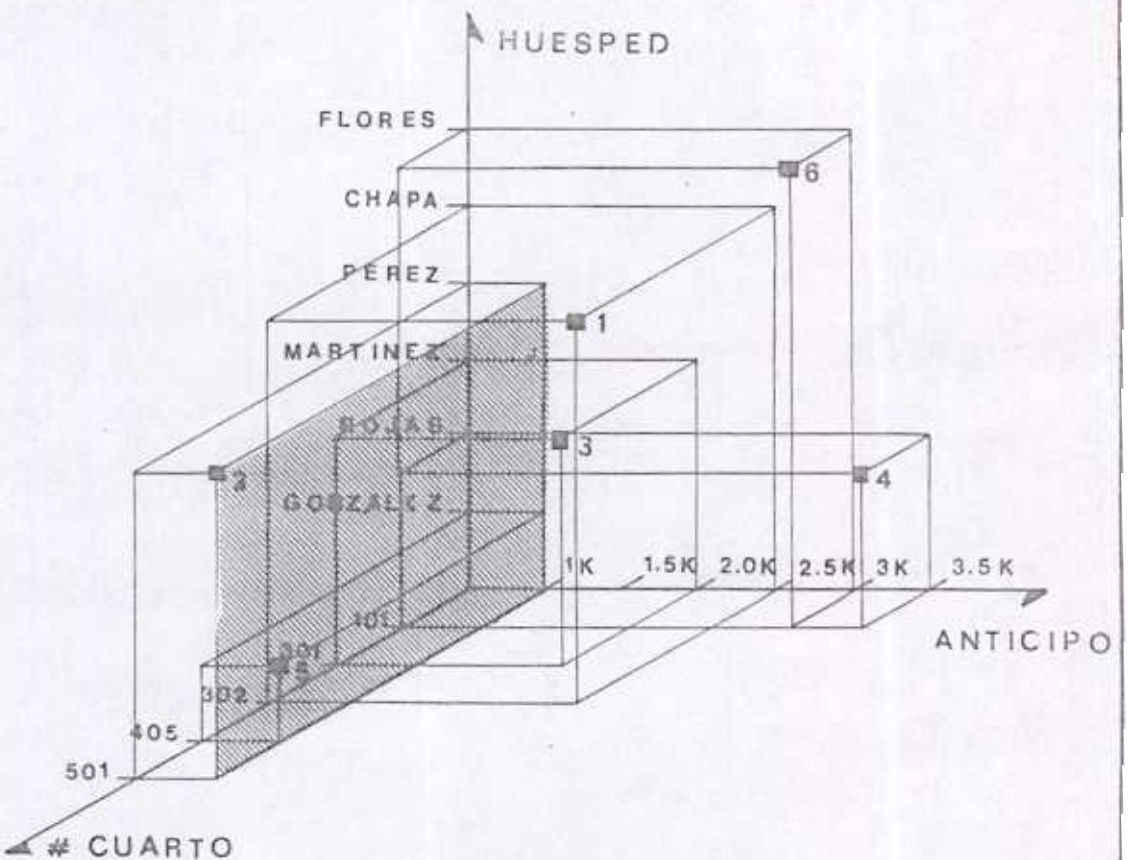
R = (HUESPED, NUMERO DE CUARTO, ANTICIPO).

En la figura 4.2 (a) designamos un eje a cada atributo para tener una representación gráfica de la relación en términos del producto cartesiano, en donde los dominios son conjuntos de diferentes tipos. En cada vertice numerado de 1 al 6 tenemos tuplas de la relación, que se representa en una tabla en la figura 4.2 (b).

Dom (HUESPED = (Cadena de letras de a lo mas 10 caracteres)
(Chapa, Perez, Martinez, Saldana, Gonzalez)

Dom NUMERO DE CUARTO) = (Numeros enteros de tres digitos)
(301, 201, 405, 103, 102, 505)

Dom (ANTICIPO = (Numeros de Punto Flotante)
(1 000.00, 2 500.00, 3 000.00, 4 000.00, 5 000.00).



a) Grafica de Relación

ESQUEMA DE RELACION	RESERVACION		
	HUESPED	# CUARTO	ANTICIPO
1	CHAFA	302	2.5 K
2	PEREZ	501	1K
3	MARTINEZ	301	2.K
4	ROJAS	101	3.5 K
5	GONZALEZ	405	1K
6	FLORES	101	3K

TUPLAS

b) Tabla de Relación

FIG. 4.2 ESQUEMAS DE RELACION

DESCRIPTOR D ARCHIVOS.

Un renglón importante en el desarrollo del sistema LIDA fué la clasificación de los objetos en términos de la implementación y las operaciones, se consideran las clase de acuerdo con su almacenamiento físico y el código de procesamiento. Para poder resolver la correspondencia en la definición de objetos en el nivel conceptual y el esquema descriptivo, se definen operaciones respecto al primero para ser implementadas con una clase de herramienta respecto al segundo. Gran parte del valor de las técnicas usadas en SBD consideran definiciones de estructuras de archivo para la integración de los datos; los elementos de datos son agregados para conformar registros y estos a su vez se conforman datos más elaborados constituyendo archivo (Fig. 4.3).

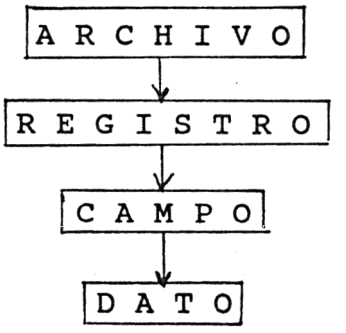


FIG. 4.3 AGREGADO EN
ESTRUCTURAS DE ALMACENAMIENTO

Con la finalidad de resolver la transición del modelo conceptual a las estructuras de almacenamiento, el sistema se vale de un esquema intermedio que documenta las estructuras y permite los accesos a la base de datos.

El sistema consta de un "Descriptor de Archivos" [CHAP85] transparente al usuario, y mediante el lenguaje de definición de datos -también con enfoque gráfico- se especifican los elementos datos manteniendo una estructura de acceso. Los mecanismos de acceso se dán a través de los procedimientos con un lote de funciones primitivas llamadas VAL para almacenar y recuperar entidades instanciadas en la base de datos.

El descriptor de archivos es implementado en LIDA como un esquema que permite el acceso a los elementos de datos de un archivo de una forma sencilla y eficiente. Los descriptores de archivos contienen una descripción de las caracteristicas físicas del archivo, la estructura de datos, la identificación de cada uno de los atributos y la descripción de cada uno de ellos.

Un descriptor de archivos D es un registro constituido de un

NOMBRE DEL ARCHIVO	NUMERO DE REGISTROS	NUMERO DE PAL. / REG.	TIPO DE ARCHIVO	TAMAÑO DE LOS CAMPOS DE DESCRIPTORES	NUMERO DE ELEMTOS

NOMBRE DEL ELEMENTO	APUNTADOR INICIAL	APUNTADOR FINAL	TIPO DE VARIABLE

DESCRIPTOR GENERAL DE ARCHIVOS

TABLA 1

EMPLEADO	125	35	S	S	13	NOMBRE	1	30	A	RFC	31	40	A	

DESCRIPTOR DEL ARCHIVO EMPLEADOS

TABLA 2

PERFIL	2500	58	D	S	14	CLAVE	1	3	1	DEPARTAMENTO	4	18	A	

DESCRITOR DEL ARCHIVO DE PERFIL

TABLA 3

TABLA	5300	32	D	5	6	NUMERO DE FACTURA	1	8	1	SUBTOTAL	9	24	P	

DESCRIPTOR DEL ARCHIVO DE TABLA

TABLA4

NOMINA	2000				4	CLAVE	1	3	1	FALTAS	6	8	1	

DESCRIPTOR DEL ARCHIVO DE NOMINA

FIG. 4.4 DESCRIPTORES DE ARCHIVOS

cierto número de bytes determina la semántica de los datos que se encuentran en los registros de los archivos. La asociación se considera a través de las descripciones del archivo de datos y apuntadores que determinan en donde se encuentran los datos dentro de los registros del archivo en cuestión. En el descriptor se tiene la identificación de los atributos para que sean asociados con los valores que tiene un tipo y se encuentran en los archivos que se estan describiendo.

Los descriptores son agrupados en un conjunto de registros para conformar un archivo denominado Archivo de Descriptores (Fig. 4.4). Un archivo de descriptores esta constituido de una secuencia de registros, en donde cada registro determina un descriptor para un archivo de datos. El archivo de descriptores tiene la misma configuración en sus registros, pudiendo ser recuperados cada uno de ellos con la finalidad de sacar el nombre del archivo de datos y el elemento que se requiere.

4.3 MECANISMO DE ACCESO A LA BASE DE DATOS.

Una de las tareas principales del sistema LIDA consiste en determinar un conjunto de valores que se encuentran en los archivos de datos para efectuar recuperaciones, modificaciones y actualizaciones impuestas por transacciones que se llevan al cabo en la base de datos. El proceso de instanciación de los datos es jerárquico descendente.

OBJETO --------> ATRIBUTO ----------> VALOR

Primero, explorando el archivo de descriptores para verificar la existencia de los objetos de relación en cada uno de los registros de descriptor; en él se examina el registro para encontrar un nombre de atributo y recuperar una conjunto de parámetros, que permitan proceder a evaluar los datos que se encuentran en los registros de los archivo de datos. La recuperación se lleva a cabo a través del esquema general de acceso de archivo de descriptores (Fig. 4.5), con una función que es la más elemental permitiendo el señalamiento a cualquier dato para su recuperación, modificación e introducción de datos en los archivos. Inclusive con los cambios dinámicos que se suceden en el descriptor.

De la misma forma que los archivos de datos, en el momento de ejecucuión de una aplicación, el archivo de descriptores, se encuentra sujeto a cambios los cuales se relacionan con el tipo de transacción y resultado que espera el usuario. Al traves del descriptor como mecanismo de acceso a la base de datos, se efectuan un conjunto de acciones elementales en terminos de secuenciación, selección y repetición; para dar como resultado una transacción. Y con un conjunto de transacciones el usuario elabora un programa de aplicación con un punto de vista de abstracción procedural.

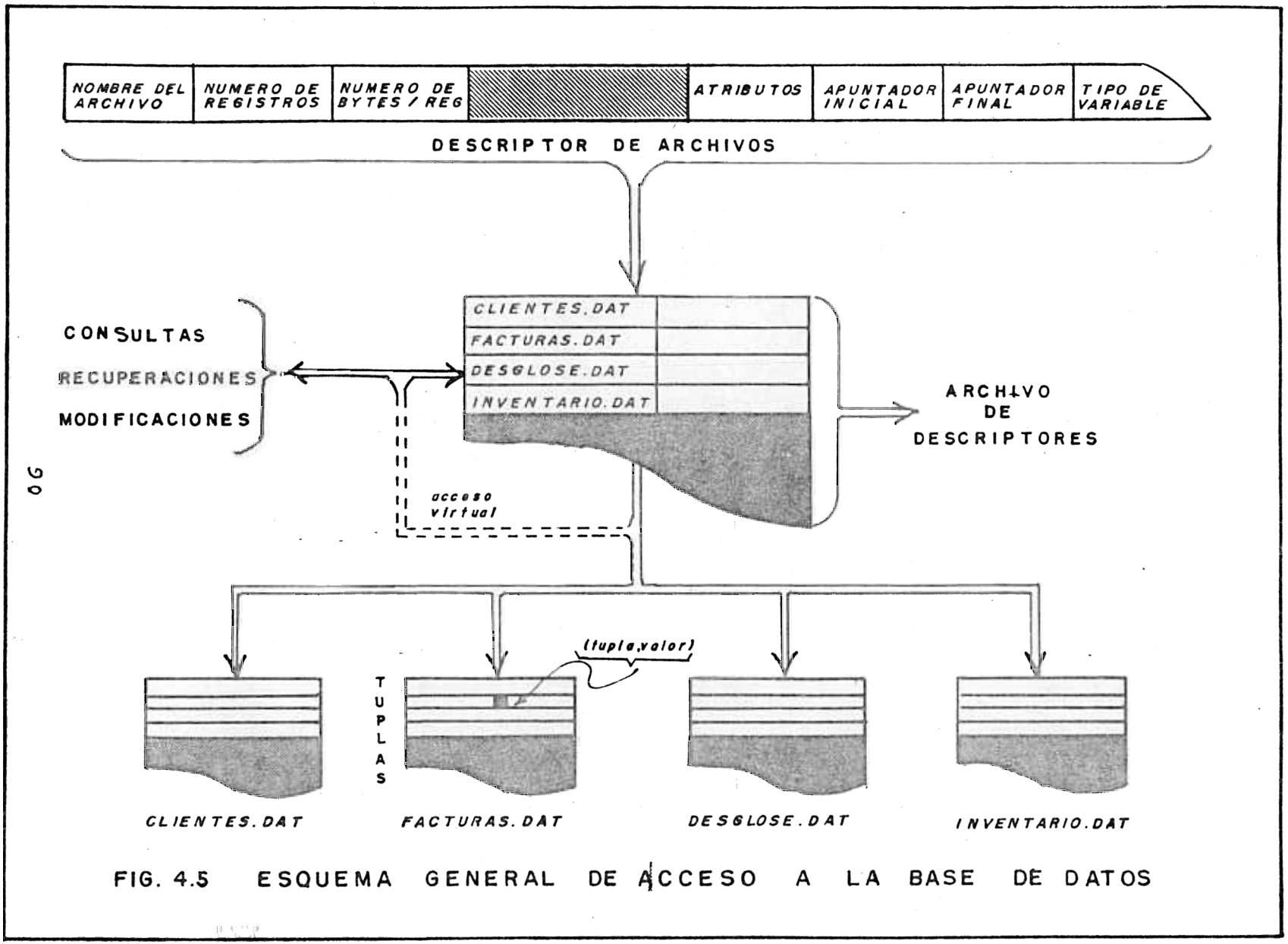


FIG. 4.5 ESQUEMA GENERAL DE ACCESO A LA BASE DE DATOS

La recuperación de los datos que se encuentran en cada uno de los registros del archivo de datos se efectua con una función elemental denominada VAL. La función VAL es el instrumento para recuperar valores vi de un conjunto de valores que se encuentran en un objeto de orden mayor que es un registro

v1,v2, .,vn).

Los valores se asocian con su identificador que es un nombre de atributo, de un conjunto de nombres que se encuentran en el descriptor

(d1,d2, ,dn).

Cada elemento df del descriptor es parte del argumento de la función VAL para tomar dos apuntadores que permiten obtener un valor vf dentro de algún registro del archivo de datos. En general, dado uno de los elementos de los descriptores di pueden considerarse con varias especificaciones asociadas a un valor de un registro contenido en un archivo de datos.

La función es una herramienta de "software" que nos permite recuperar valores de los archivos de acuerdo a un elemento di del descriptor de archivos. VAL toma como argumento el registro y el elemento de descriptor para que a través de los apuntadores la función pueda recuperar el dato de una forma precisa (Fig. 4.6). El lenguaje LIDA requiere de una serie de llamados de la función VAL para poder modificar el estado de un objeto de datos en el cálculo de procedimientos.

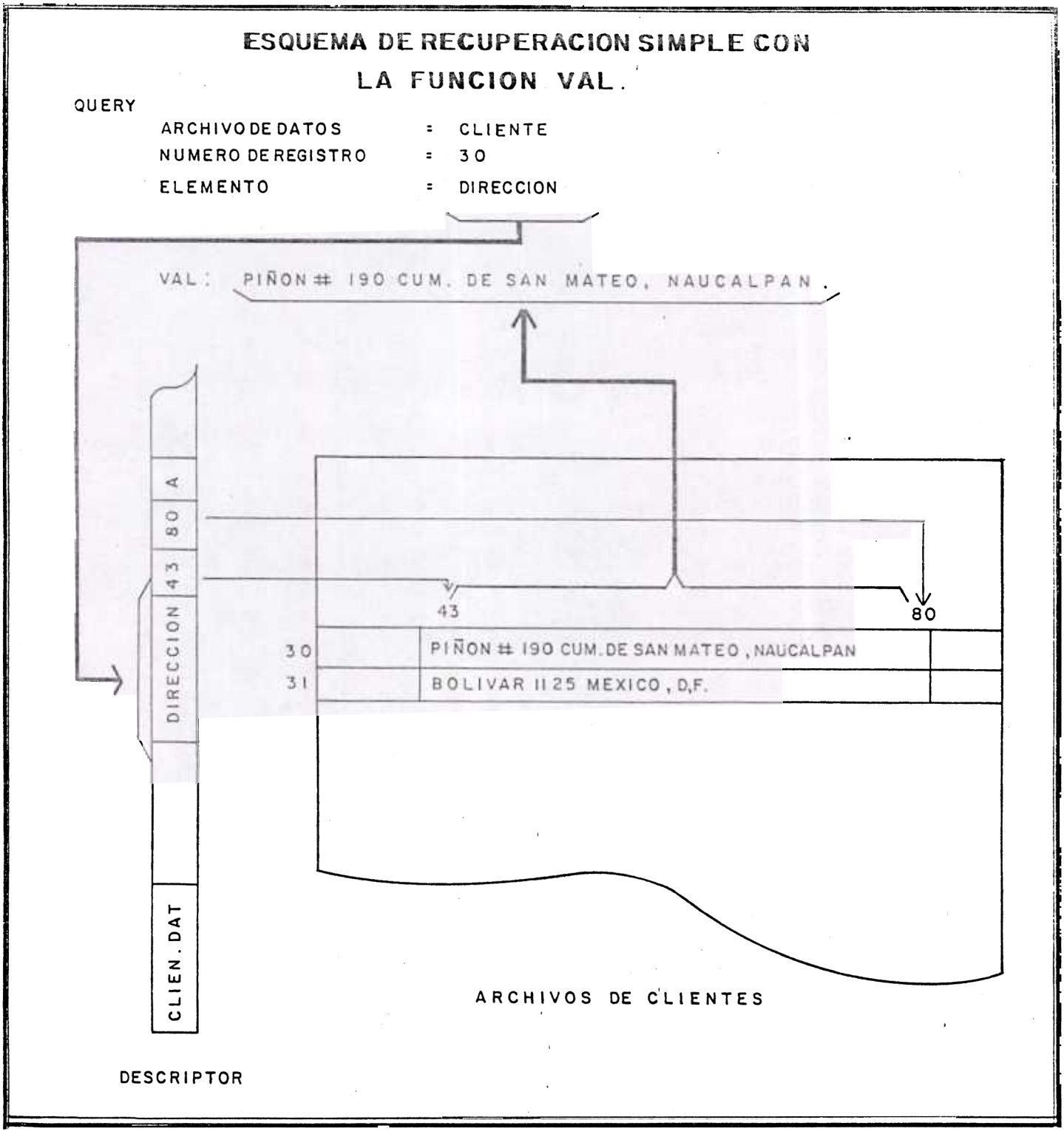


Fig. 4.6 Esquema de Recuperación Simple con la Función Val.

4.4 LENGUAJE DE DEFINICION DE DATOS DE LIDA

En general los sistemas manejadores de base de datos tienen un subsistema que soporta en Lenguaje de Definición de Datos (LDD). Por ende, los sistemas dedicados al desarrollo de aplicaciones que estan provistos de una base de datos amplia, deben tener un subsistema similar como los ejemplos presentados en el capítulo dos y tres: BDL que tiene el subsistema "Form Definition Component"; SBA que usa QBA como un lenguaje visual que le permite la definición de datos; y FORMAL que proporciona al usuario su componente de especificación de formas. Cada uno de los lenguajes tiene como proposito de mostrar una vista conceptual de acuerdo a los objetos de datos del mundo real.

El LDD tiene por objetivo la definición de los esquemas y subesquemas de las bases de datos, que planean los tipos de entidades y las formas cómo se vinculan de la definición en nivel conceptual a una representación de estructura de máquina. El sistema de LIDA contiene un LDD que se encarga de definir el esquema conceptual y el esquema físico en terminos del módulo Descriptor y el mismo Descriptor de Archivos. Podemos decir que el LDD de LIDA es estilo gráfico manejando fundamentalmente los objetos de datos, con el despliegue de ventanas en la pantalla para ser llenadas en esqueletos de formas tabulares.

El siestema LIDA para definir y construir su base de datos tiene dos componentes principales: el Descriptor y el Captador que aparecen como opciones en el menú de la primer pantalla para ser usados indistintamente con solo poner el cursor. En el capítulo 1 se trató con detalle estos sistemas siguiendo su procedimiento de operación con una sesión en LIDA.

C A P I T U L O

DEFINICION DE LA SINTAXIS Y SEMANTICA DE LIDA

El objetivo de este capitulo definir la sintaxis y la semántica de LIDA. Las reglas sintácticas ensamblan figuras geométricas con entradas y salidas que determinan implicitamente el control de flujo. Los componentes principales del lenguaje LIDA son iconos que representan procesos e iconos flechas que establecen el flujo de datos, para formar un flujograma con sintáxis bidimensional haciendo patente una exposición paralela de los procesos. En LIDA los iconos contienen una semántica determinada por el comportamiento de los operadores de alto orden o transformaciones sobre relaciones.

La idea principal de este capitulo es mostrar como la abstracción procedural se determina intrinsecamente en LIDA con la semántica de los operadores y el flujo de datos que especifica el control de flujo, aspectos relevantes en los lenguajes de muy alto nivel. La abstracción procedural se relaciona con la construcción de procesos y la definición de operaciones, comportamiento de los objetos que se liga estrechamente con la parte estructural del modelo de abstracción de datos. En LIDA la especificación del control de flujo se relaciona directamente con el modelo de flujo de datos y el constructor condicional que resuelve el problema de bifurcación en un flujograma equivalente a la estructura,

IF <condición> THEN <postulado>
ELSE <postulado>

que ocurre en un programa.

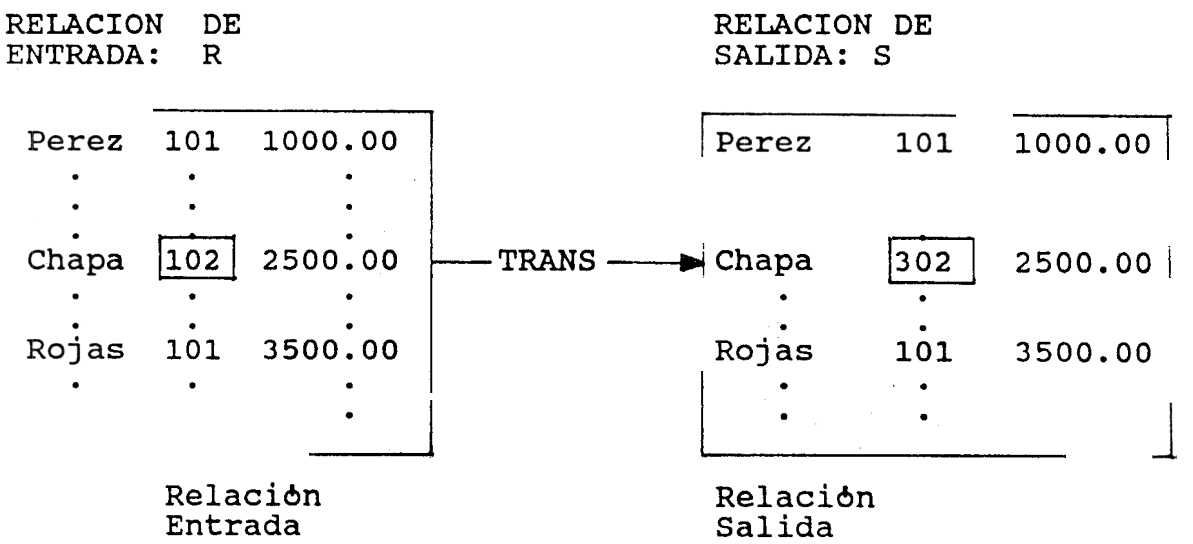
Se considera la denominación del procedimiento y el punto de llamado como una caja negra la cual regresa un valor o lleva al cabo algunos cambios de estado en la máquina. En este nivel, la subrutina tiene algún efecto observable pero no se tiene el detalle de como se realiza. En abstracción procedural las piezas de "software" se construyen con orientación funcional , para que de esta forma se usen y se ensamblen en programas complejos.

5.1 PRINCIPIOS DE DISEÑO EN EL LENGUAJE LIDA.

La abstracción procedural en LIDA consiste en el encapsulamiento de macros en iconos que definen operaciones sobre objetos de datos. Los macros en un contexto de generalización son abstracciones sin considerar los detalles del procedimiento, para dar pie a la solución de problemas a la medida de casos particulares. Por ejemplo, la abstracción de actualización, considera un proceso que tiene en su entrada un archivo maestro y otro de transacciones para dar como resultado un archivo actualizado. Sin embargo, para llevar a cabo el proceso de actualización

es necesario particularizar en detalles la transacción con los métodos de añadir, borrar o cambiar elementos, como se muestra en la figura 5.2.

La abstracción procedural en el diseño de un lenguaje genera un compromiso entre generalidad y flexibilidad; con abstracciones que algunas veces suelen ser singulares pero poco representativas y algunas otras muy generales que requieren ser detalladas en su aplicación o dificilmente de usarse en una implementación. Partiendo del caso más singular (figura 5.1), a un objeto aplicamos una transformación que consiste en cambiar solo un valor, que se presenta en un dominio de atributo y una eneada:



5.1 Transformación Simple de una Relación

Con una representación en pseudocódigo de la siguiente forma:

```
     PROCEDURE TRANS;
     { Cambia la asignacion de numero de cuarto a Chapa

     IMPORTA Relacion R;
     EXPORTA Relacion S;

     i=0
     REPEAT 1;
     GET (Registro (i), R);
     (Nombre, #Cuarto, Anticipo) = Registro (i)
     IF < Nombre .EQ. CHAPA > THEN #_Cuarto = 302;
     WRITE ( registro(i), S);
     i:= i+1
  1  UNTIL not <EOF>;
     END.
```

La transformación anterior de cambiar un solo valor es la más singular; sin embargo, en la elaboración de programas de aplicación requerimos de una serie de transacciones que de forma sistemática suelen afectar a un mayor número de elementos, tanto en los dominios como en las eneadas. En la figura 5.2 mostramos como se afecta la estructura de esquema de relación cuando llevamos a cabo alguna de las operaciones básicas de inserción, adición y borrado. En esta misma figura puede observar la definición de una función sobre una relación. La evaluación se lleva a cabo sobre el dominio del atributo edad, considerada como la función promedio PROM.

Los iconos son transformaciones u operadores que actúan sobre relaciones para dar como resultado relaciones, con excepción de la función. En general, de cada icono pueden salir varias líneas de flujo con la misma relación, de tal forma que puede ser utilizada como entrada para diferentes iconos. La excepción es el icono de escritura de relación y variable que son terminales en un flujograma. La sintáxis para LIDA se establece con el icono y entradas y salidas:

*Iconos iniciales.- Son de lectura y sólo salen líneas de flujo de datos

*Iconos finales.- Son de escritura y sólo admiten una línea de flujo de datos de entrada.

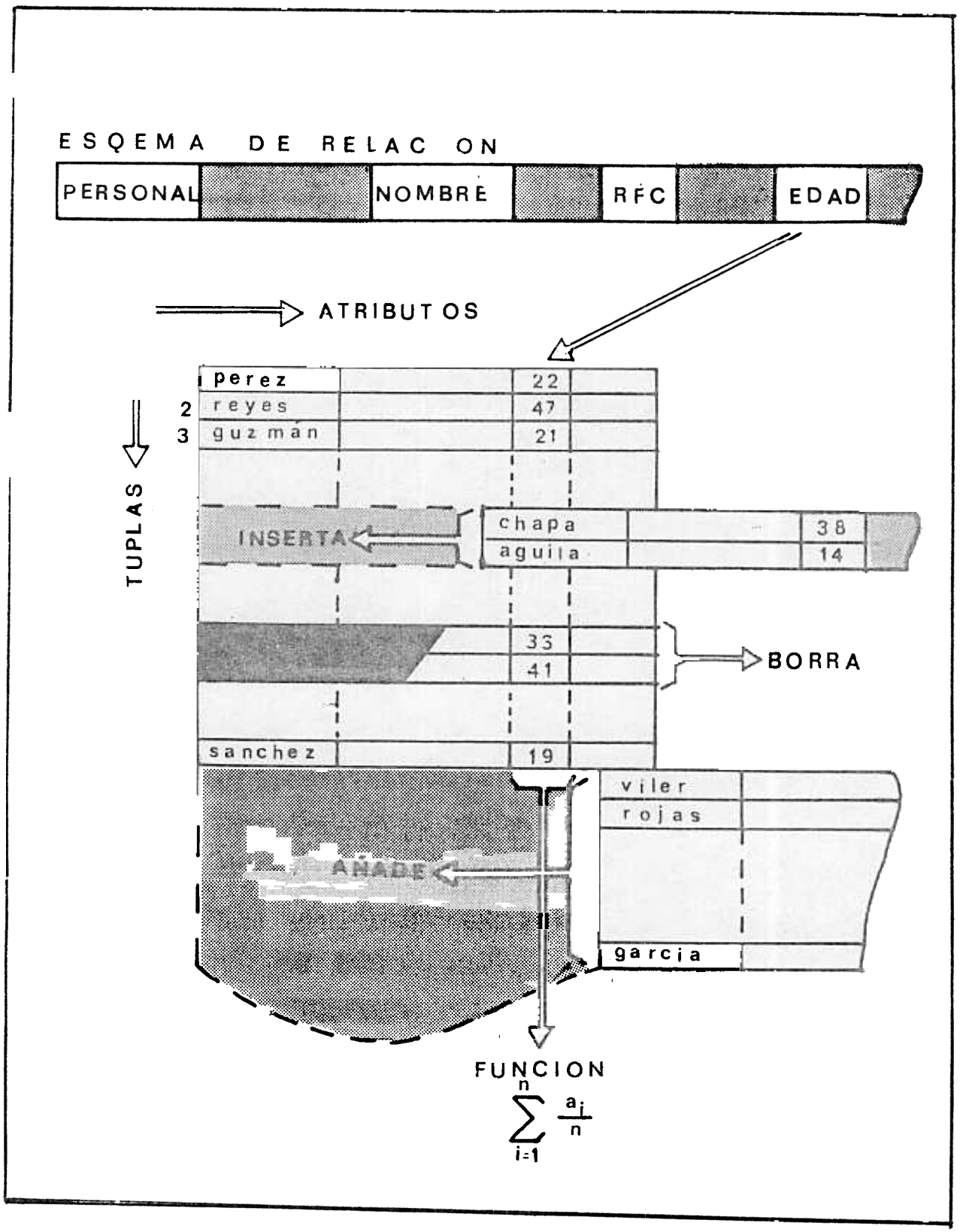
*Iconos tipo operador binario.- Acepta dos líneas de flujo de datos como entrada. Una o varias líneas de flujo de salida.

*Iconos tipo operador unario o proceso.- Acepta una línea de flujo de datos como entrada y una de salida. Tiene argumentos que son atributos y expresiones algebráicas.

*Iconos constructor condicional.- Acepta una línea de flujo de datos de entrada. Salen dos líneas de flujo de datos distinguibles. Tiene como argumento una expresión booleana.

*Iconos función.- Acepta una línea de flujo de entrada y sale una sóla línea de flujo que porta un valor que se almacena en icono rectángulo con un nombre.

En LIDA aplicamos el principio de ortogonalidad con la definición de un conjunto de operadores que fueran lo suficientemente generales e independientes para poder desarrollar programas de aplicación en el procesamiento de datos. En LIDA se define en seis clases fundamentales de iconos relacionados con su naturaleza transformacional u operativa. Todos están unificados con el modelo relacional, debido a que operan sobre relaciones y producen relaciones, excepto para las funciones en los que producen solo valores.



g 5.2 Transformaciones de Actualización.

* Iconos de Entrada y Salida de objetos de datos
* Iconos para la Transacción de objetos de datos
* Iconos de Operadores Relacionales.
* Iconos de Funciones.
* Iconos de Procesos.
* Iconos de Construcción de Condicionales.

La elaboración de flujogramas en LIDA como programas de desarrollo de aplicaciones se lleva a cabo con una secuencia de transformaciones en donde cada una de ellas tiene una entrada de objetos de datos y una salida; de tal forma que la primer clase de iconos son los que representan procesos de lectura y escritura de los objetos de datos. La diferencia esencial entre uno y otro es la salida de líneas de flujo de datos y en el otro caso la entrada de solo una línea de flujo de datos respectivamente.

La segunda clase de iconos corresponde a los que tienen asociado un proceso algoritmico muy preciso, común en los programas que transforman archivos. Este tipo de procesos son los que regularmente se establecen como operadores genéricos dentro de los lenguajes de manejo de datos y desarrollo de aplicaciones, correspondiendo a procedimientos especificos. Es posible aumentar el número de iconos con procedimientos que resuelvan algunos problemas especificos.

La tercer clase de iconos corresponde a la representación de los operadores del algebrá relacional, con una primer subclase de operadores booleanos de unión, intersección, y diferencia; basados en la teoría de conjuntos. La definición de las operaciones se establece con los mismos esquemas de relación, siendo inrelevante especificarlos dentro de los iconos como atributos de restricción.

La proyección es un operador unario con atributos de restricción que se incorporan en el icono, determinando los dominios que son proyectados a una nueva relación.

Uno de los iconos más usado en los flujogramas son los correspondientes a la clase de operadores denominada como junta natural o junta con operador. Dicha operación junta dos relaciones en base a dos pivotes que son dos atributos comunes a las relaciones. Sobre un dominio para cada valor que representa una tupla se compara con cada uno del otro dominio para establecer una junta de tuplas de las dos relaciones cuando se cumpla la condición de igualdad o desigualdad establecida. En el caso de la junta natural se consideran los nombres de los atributos que son iguales.

La cuarta clase de icono representa funciones que evalúa de una relación sobre un dominio determinado por un atributo de restricción. En este caso se tiene como entrada una relación y la salida es un valor representada por un parámetro. Actualmente tenemos funciones bien determinadas como máximo, mínimo, suma y cuenta. Sin embargo, es posible definir nuevas relaciones mediante la especificación de una fórmula que involucra variables

que se instancian solo en el dominio de restricción (por ejemplo: varianza, promedio, etc.).

Los iconos que representan procesos son una quinta clase con amplias posibilidades para el procesamiento de datos. En el proceso la especificación del atributo denota en donde se va a depositar el resultado del proceso que se lleva a cabo sobre la relación de entrada. En los iconos de proceso se definen simbolicamente fórmulas algebráicas que se evaluan con una instanciación de valores sobre los dominios de los atributos. El proceso acepta una relación de entrada con su evaluación de forma algebráica cuyos parámetros son los atributos. El resultado es una nueva relación especificado con el nombre de otro atributo o la actualización de alguno existente.

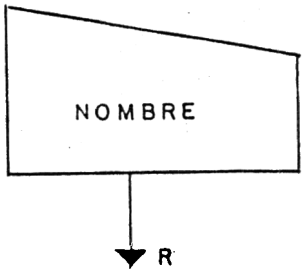
La sexta clase de icono corresponde al constructor condicional el cual siempre espera una relación de entrada y produce dos relaciones de salida, de acuerdo a una bifurcación definida por el valor verdadero o falso. En LIDA es posible construir condicionales compuestos mediante los conectivos de conjunción y disyunción. Este caso es similar al anterior, debido a que es necesario definir una fórmula lógica para ser posteriormente evaluada durante la ejecución.

En resúmen, en LIDA los iconos importan relaciones a través de la línea de flujo de entrada para procesarlas con argumentos de restricción que son atributos, fórmulas lógicas y métodos; para exportar nuevas relaciones sobre las líneas de flujo de datos. De la misma manera se definen iconos que representan procesos y funciones que se especifican con formas algebráicas; con la excepción que el último solo exporta valores.

5.2 ICONOS DE LECTURA / ESCRITURA DE OBJETOS EN EL LENGUAJE.

ICONO LECTURA DE RELACION

<SINTAXIS>



LECTURA DE OBJETO

<REPRESENTACION EN CADENA>

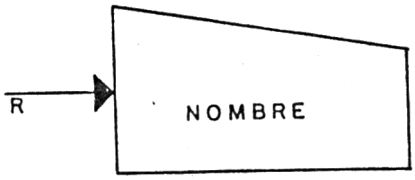
LEE R

<SEMANTICA>

El simbolo LECTURA significa leer una relación cuyo nombre se especifica dentro del icono. Se consulta al descriptor de archivos y se ve si la relación esta dada de alta.

ICONO ESCRITURA DE RELACION

<SINTAXIS>



ESCRITURA DE OBJETO

<REPRESENTACION EN CADENA>

ESCRIBE R

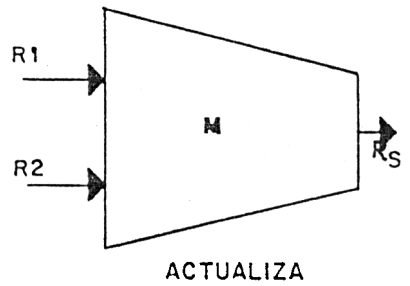
<SEMANTICA>

El simbolo significa escribir una relación en el archivo. El icono tiene dentro el nombre de la relación que se va a dar de alta en el descriptor.

5.3 ICONOS QUE REPRESENTAN TRANSACCION DE RELACIONES.

ICONO ACTUALIZA

<SINTAXIS>



<REPRESENTACION EN CADENA>

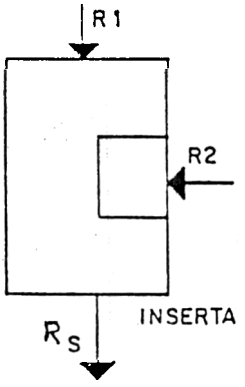
```
R1   R2 [ M , Key   = Rs

|  Actualiza
M  Método  A: Añade
           B: Borra
           C: Cambia
Key = Llave
```

<SEMANTICA>

El icono actualiza significa modificar una relación de entrada R1, con otra relación de transacciones R2. La relación se altera con el agregado de nuevas tuplas de la relación R2 (método Añadir). Borra algunas tuplas determinadas por atributos llave ordenados en una relación unaria R2. Cambia un grupo de tuplas determinadas en la relación de transacción R2, localizadas con el empatamiento de un atributo llave. El icono se conecta con dos líneas de flujo de datos de entrada correspondiente a la relación R1 y R2; Otra línea de flujo de datos de salida apunta a otro icono. Dentro del icono se pone el método correspondiente: Añade, Borra y Cambia.

ICONO INSERTA

<SINTAXIS>



<REPRESENTACION EN CADENA>

R1 ^ R2 [Key] = Rs

Inserta

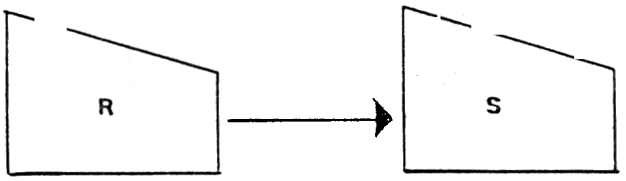
<SEMANTICA>

El icono inserta expresa una inclusión de una relación en otra. La inserción se lleva a cabo a partir del registro especificado como argumento. Considera un archivo definido por el descriptor de archivos y otro archivo que tiene el mismo número de columnas y atributos iguales. A partir de donde se da el número de registro se incluye un archivo en otro. Dá como resultado otro nuevo archivo que se define en el descriptor de archivos.

EL icono se ensambla dentro del flujograma con dos líneas de flujo de datos que apuntan al símbolo gráfico, el resultado es un nuevo objeto de salida con otra línea de flujo de datos. Una linea de flujo de información incide sobre un lado del icono, sucediendo en la parte lateral otra línea de flujo de datos de entrada. El icono exporta una relación a través de una línea de flujo de salida opuesta a la línea de entrada.

ICONO COPIA

<SINTAXIS>



COPIA

<REPRESENTACION EN CADENA>

R = S

= Copia

<SEMANTICA>

Se tiene una relación de entrada que se importa para copiarla en una relación de salida con destino al icono de escritura con el nombre dentro del icono de escritura que identifica la nueva relación, en este caso S. Se agrega un registro en el descriptor de archivos del archivo determinado con las características del fuente. Este es un caso particular de proyecta.

ICONO ORDENA

<SINTAXIS>

ORDENA DES

ORDENA ASC

<REPRESENTACION EN CADENA>

R # Atributo, A = S R # Atributo, D = S

Ordena

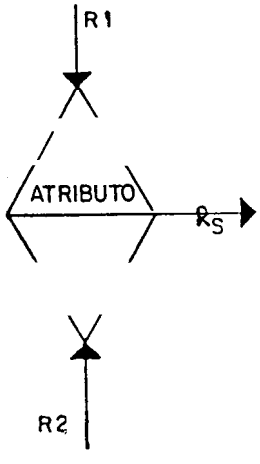
<SEMANTICA>

El significado del icono es el de ordenar en forma ascendente o descendente las tuplas de la relación. El ordenamiento se lleva a cabo conforme a los valores del dominio de un atributo que se determinón cómo argumento del icono. El método de ordenamiento ascendente o descendente esta dado por la diferenciación de los iconos. El resultado es un objeto ordenado en forma ascendente o descendente bajo el atributo especificado, según el caso que se especifica.

El icono admite fundamentalmente dos líneas de flujo de información; una de entrada de datos y otra de salida de ellos.

ICONO MEZCLA

<SINTAXIS>



MEZCLA

<REPRESENTACION EN CADENA>

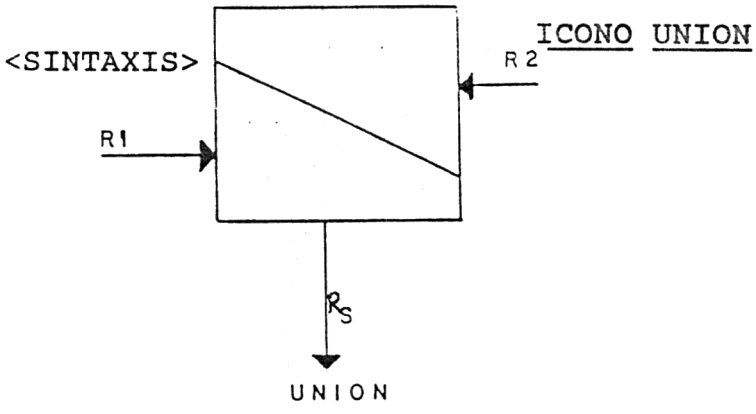
R1 @ R2 [Atributo = Rs

@ Mezcla

<SEMANTICA>

El significado del icono mezcla se intercalan los registros de dos archivos para generar un nuevo archivo ordenado de forma descendente. El operador mezcla toma valores sobre el dominio de los objetos con un atributo seleccionado. Las dos relaciones deben tener el mismo esquema de relación con la existencia de una llave en común que es el atributo que usamos para intercalar las tuplas. En el icono mezcla se tienen dos líneas de entrada para generar un resultado en una línea de flujo de datos de salida.

5.4 ICONOS QUE REPRESENTAN OPERADORES DEL ALGEBRA RELACIONAL



<REPRESENTACION EN CADENA>

R1 + R2 = Rs

+ Unión

<SEMANTICA>

Dos relaciones sobre el mismo esquema de relación puede considerarse como dos conjuntos sobre el mismo universo, lo cual puede interpretarse como el conjunto de todas las posibles tuplas sobre el esquema de relación. De esta forma definimos el icono que representan la unión de R1 y R2, como el conjunto de todas las tuplas que están en R1 y también en R2.

Ejemplo:

R1	A	B	C
	a	b	c
	d	a	f
	c	b	d

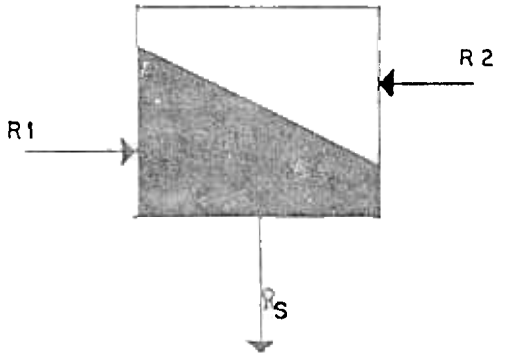
R2	A	B	C
	b	g	a
	d	a	f

R1+R2	A	B	C
	a	b	c
	d	a	f
	c	b	d
	b	g	a

El icono de unión representa un operador binario que tiene dos lineas de flujo de datos por donde fluyen dos conjuntos de tuplas definidas sobre el mismo esquema para generar un resultado con el mismo esquema y salida en una línea de flujo que conecta con algún otro icono.

ICONO DIFERENCIA

<SINTAXIS>



DIFERENCIA

<REPRESENTACION EN CADENA>

R1 - R2 = Rs

Diferencia

<SEMANTICA>

El significado del icono diferencia se asocia al operador que construye el conjunto de todas las tuplas que se encuentra en la primera relación menos las que se encuentra en la segunda relación. Las dos relaciones que se importan están definidas sobre el mismo esquema de relación para dar un resultado dentro de mismo esquema. El resultado que fluye por una línea de salida es un conjunto de tuplas que están en R1 pero que no son comunes a R2.

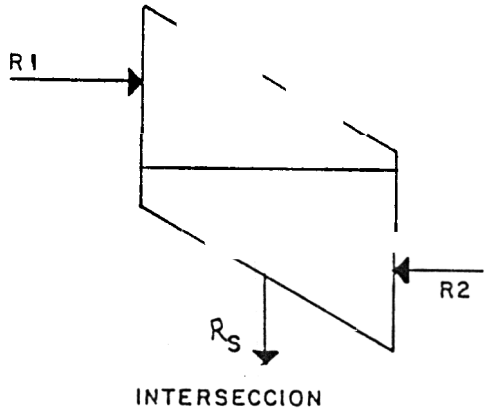
Ejemplo: Considere R1 y R2 cómo en el ejemplo anterior. El resultado es

R1-R2	A	B	C
	a	b	c
	c	b	d

El icono diferencia es similar al de unión salvo que la relación que se quiere substraer apunta a una región de mayor intensidad de luz dentro del icono. De la misma forma que la unión, el icono diferencia se ensambla en un flujograma con dos líneas de entrada que inciden sobre el icono en sus lados laterales opuestos, para derivar una línea de salida que porta el resultado.

ICONO INTERSECCION

<SINTAXIS>



<REPRESENTACION EN CADENA>

R1 . R2 = Rs

Intersección

<SEMANTICA>

La semántica del icono intersección es la que corresponde a la intersección de dos conjuntos. Las dos relaciones que se importan como argumentos al operador están definidas sobre el mismo esquema de relación obteniendo todas las tuplas que se encuentran en R1 y también en R2. El icono intersección representa un operador binario con dos líneas de flujo de entrada y su resultado en una de salida.

Ejemplo.- Sean R1 y R2 las siguientes relaciones:

R1	A	B	C
	a	b	c
	d	e	f
	f	g	h
	e	a	b
	d	f	g

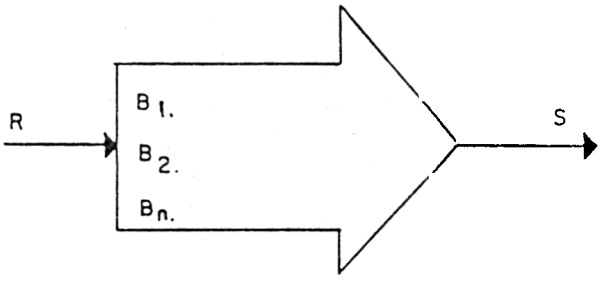
R2	A	B	C
	d	f	g
	e	a	b
	a	c	b

entonces la intersección es:

R1.R2	A	B	C
	d	f	g
	e	a	b

<SEMANTICA>

ICONO PROYECTA



PROYECCION

<REPRESENTACION EN CADENA>

R % A1,A2, ..., An = S

% Proyecta

<SEMANTICA>

El icono proyecta encapsula el operador relacional que consiste en escoger un subconjunto de columnas determinadas por los nombres de los atributos que se encuentran escritas dentro del icono. El icono importa una relación con la línea de flujo de entrada, generando una salida de relación que es un subconjunto de la relación de entrada.

Ejemplo.- Sea R1 una relación

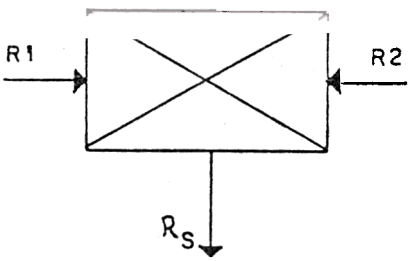
R1	A	B	C	D
	c	b	e	f
	a	c	g	h
	d	e	f	g

R1% A,B

A	B
c	b
a	c
d	e

ICONO JUNTA NATURAL

<SINTAXIS>



JUNTA NATURAL

<REPRESENTACION EN CADENA>

R1 * R2 = Rs

<SEMANTICA>

El icono junta natural encapsula un operador binario el cual importa objetos de relación por médio de dos líneas de flujo de datos. El icono es aplicable solamente cuando las relaciones tiene al menos un atributo denominado de la misma manera y los tipos corresponden. El significado es el de seleccionar todas las tuplas cuyos valores en la columna del atributo A de la relación R1 concuerden con los del atributo igual B de la relación R2. Las tuplas se juntan una a una sin duplicar el atributo que es común, generando una nueva relación.

Ejemplo.- Sean R1 y R2 definidas como:

R1

A	B	C
a	b	c
e	f	g
m	n	d
h	j	c

R2

C	D	F
d	f	g
c	a	h
a	b	c

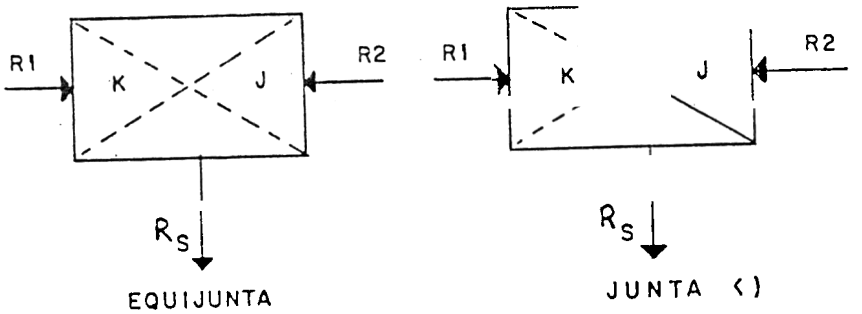
R1 * R2

A	B	C	D	F
a	b	c	a	h
m	n	d	f	g
h	j	c	a	h

El icono de junta natural se ensambla con tres líneas de flujo de datos. Dos de entrada que sirven como argumento para alimentar el proceso y una de salida que puede ser usada por otro proceso. Las dos líneas de entrada se disponen sobre los lados laterales y el resultado se obtiene con la línea de flujo que sale sobre el lado adyacente.

ICONO JUNTA CON OPERADOR

<SINTAXIS>



<REPRESENTACION EN CADENA>

R1 *= R2 [Ai,Aj = Rs

*= EQUIJUNTA

R1 *< R2 [Ai,Aj] = Rs

*< Junta con desigualdad

<SEMANTICA>

En el icono junta con operador concurren dos lineas de flujo de entrada que representan los dos argumentos de relación R1 de orden k y R2 de orden m. El significado del icono consiste en comparar valores del atributo Ai correspondientes a la relación R1 con los valores del atributo Aj de la relación R2, para juntar las tuplas que cumplan con la condición. El proceso consiste de dos ciclos anidados en donde se toma cada valor de la columna i, para compararlo segun el operador (=, <), con cada valor de la columna j de R2. Los valores que cumplan con la condición determinan la selección de las correspondientes tuplas juntandolas en una nueva tupla de orden k + m. El proceso se lleva a cabo para cada tupla de R1 contra todas las tuplas de R2.

Ejemplo.- Sea R1 y R2 las siguientes dos relaciones:

R1

A	B	C
1	2	3
4	5	6
7	8	9

R2

D	E
3	1
6	2

El resultado de R1 *< R2 [B,D]

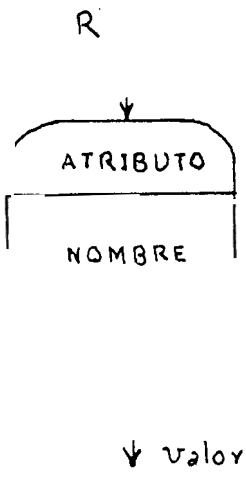
A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

El caso de la equijunta es similar tomando en cuenta que se toma la igualdad con los atributos que se quieren comparar. El icono tiene dos lineas de flujo de datos como entradas y una de salida.

5.5 ICONOS QUE REPRESENTAN FUNCIONES Y PROCESOS.

ICONO FUNCION

<SINTAXIS>



<REPRESENTACION EN CADENA>

R FUNCION [Relación, Atributo = Valor

MAX Máximo
MIN Mínimo
PROM Promedio
SUM Suma
REGS Número de registros

<SEMANTICA>

El icono función es genérico para representar diferentes funciones, lo único que cambia es el nombre determinado por una cadena de cuatro letras que se especifica dentro del icono. El icono función tiene como argumento la relación de entrada R y el dominio sobre el cual se evalúa la función. El resultado de este icono es un valor que se manda a una línea de flujo de dato pero que sólo es recibida por un icono rectángulo que tiene especificado dentro un nombre. Este parámetro puede ser usado en el resto del flujograma, con sólo llamarlo con el nombre, por ejemplo en expresiones algebráicas.

El significado de cada función depende de cada una de ellas:

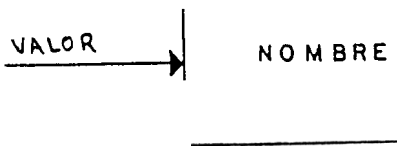
MAX Obtiene el máximo de los valores sobre un dominio.
MIN Obtiene el mínimo de los valores sobre un dominio.
PROM Obtiene el promedio de todos los valores sobre el dominio.
SUM Suma todos los valores del dominio.

REGS Cuenta el número de tuplas que existen en el grupo
En este caso se omite el nombre del atributo.

El caso del icono función acepta una línea de flujo de datos de entrada para evaluar la función sobre el atributo especificado, y mediante otra línea de flujo de salida se trasmite el valor al icono asigna variable.

ICONO ASIGNA VARIABLE

<SINTAXIS>



ASIGNACION VALORES

<REPRESENTACION EN CADENA>

@X = VALOR

<SEMANTICA>

El significado del icono es el de alojar el valor del resultado de una función en una localidad de memoria con el nombre de una variable. El valor puede ser usado con sólo usar el nombre de la variable en algún otro punto del flujograma. Por ejemplo una expresión algebráica en un proceso o una condición en un icono de selección.

El icono es terminal y es el único que acepta una línea de flujo de datos que porta un sólo valor y no una relación cómo en los otros casos.

ICONO PROCESO

<REPRESENTACION EN CADENA>

R Expresión Algebráica] = S

[] Proceso

Ejemplo.- Relación de entrada R = EMPLEADO.
Relación de salida S = NOMINA.

EMPLEADO EXP] = NOMINA

Expresión Algebráica:
SUELDOT = SUELDOB + PRESTA - IMPUESTO/1.5 + FALTAS/30

<SEMANTICA>

La semántica de este icono esta definida fundamentalmente por la especificación algebráica que se dá al proceso. El icono importa una relación para modificar su estado bajo un proceso que cambia cada una de las tuplas bajo una especificación algebráica. En este proceso es posible incluir diversos atributos y constantes, que se incluyen en la expresión algebráica.

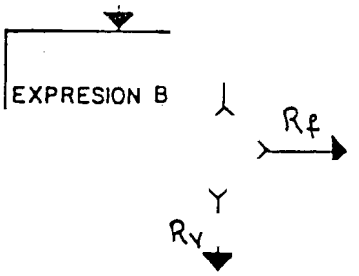
Los resultados son una nueva relación que tiene adicionado en un nuevo dominio de atributo o modificando alguno ya existente. El cálculo se hace para cada tupla con los elementos que intervienen en la forma algebráica sobre todo el dominio.

El icono de proceso tiene dos líneas de flujo de datos. Una de entrada y otra de salida. Con la primera se importa los objetos de datos, para exportarlos a otros procesos con la línea de salida.

5.6 ICONO CONSTRUCTOR CONDICIONAL.

ICONO CONDICIONAL

<SINTAXIS>



<REPRESENTACION EN CADENA>

Rv = R : (A = cte), Rf Rv = R : A < cte), Rf

Filtro Seleccción

(Expresión Booleana)

Salida Verdad
, Salida Falsa

En las expresiones booleanas podemos usar

O	"o" lógico	<	menor	<=	menor o igual
Y	"y" lógico	>	mayor	>=	mayor o igual
<>	diferente		paréntesis		

<SEMANTICA>

La semántica de este icono es la de un filtro de selección. Mediante el icono constructor condicional se produce una bifurcación en el flujo de la información en base a la evaluación de una expresión booleana que se especifica con: variables de atributo, constantes, conectivos booleanos "OR" y "AND" y operadores de comapración: igual, menor o igual y diferente.

Con los iconos condicionales se construyen dos nuevos objetos de relación con el mismo esquema a partir del flujo de datos de entrada. Los objetos de relación se bifurcan sobre dos posibles líneas de flujo de salida una verdad y otra representando una salida falsa. La construcción se efectúa con la selección de tuplas evaluando la condición de comparación para cada uno de los valores sobre el dominio del atributo.

Podemos observar que la construcción del condicional con forma lógica que involucra negación, se establece solamente con la salida falsa del constructor condicional que contiene la fórmula con operador igualdad como se muestra en la figura anterior.

Este icono tiene una línea de flujo de entrada por medio de la cual importa un objeto, produciendo dos líneas de flujo de salida considerandose como una salida verdad y otra falsa. El icono constructor condicional tiene el significado equivalente a la proposición IF <condición> THEN; ELSE. Obteniendo en este caso un control de flujo de datos.

Ejemplo: Sea R una relación de empleados con el siguiente esquema: R=(Nombre, RFC, Sueldo, Sexo, Antigüedad, Departamento). De R queremos obtener una relación S que cumpla con la siguiente condición:

EXP = (Sueldo < 1000 O Antiguedad > 20 Y Departamento "Computación"

7 EJEMPLO DE ILUSTRACION

En este ejemplo queremos presentar, en terminos de flujogramas, el desarrollo de una aplicación con su descripción de ejecución con el acceso al descriptor y cada una de las relaciones que usa y las transacciones que se van ejecutando con forma se sigue el recorrido del fluograma.

Se tiene un esquema de relación empleado que consiste en atributos de un archivo de personal y la relación tabla que contiene atributos relacionados con sueldos, percepciones y deducciones para cada categoria que tiene la compañia.

El problema consiste en encontrar ciertos valores y listar el archivo de empleados con su sueldo integrar clasificados. La finalidad es ayudar a la toma de decisiones del empresario acerca de un aumento y ajuste de sueldos. Se requiere obtener las cotas máximas y mínimas de las percepciones, así como, el promedio de las mismas. Se quiere explorar el estado de sueldos integrales con una clasificación.

La solución consiste en leer primero dos relaciones que se encuentran en los archivos de personal y tabla. En el primero se tienen todos los atributos correspondientes a datos personales, sin embargo, no se tienen los atributos de sueldos y percepciones. Cada empleado tiene una categoria que lo asocia con un conjunto de atributos que son referentes a sueldos y percepciones. Por tal razón lo primero que se lleva a cabo es juntar las tablas para tener para cada empleado sus datos de sueldo y bonos (que es el caso que estamos tratando).

Debido a que sólo requerimos de los atributos nombre, sueldo y bonos; entonces llevamos a cabo una proyección cómo puede verse en el flujograma. El resultado es una nueva relación que entra al icono proceso. En este icono encontramos para cada empleado su sueldo integral especificado cómo: sint = sueldo + bonos.

Del proceso efectuado anteriormente, se genara una nueva relación con un nuevo atributo que es el sueldo integral. De tal forma que se tiene una relación con los atributos: Nombre, Sueldo, Bonos, Sueldo Integral. De esta relación es posible responder algunas preguntas como son: el máximo y mínimo de sueldo; la suma del sueldo integral; y el promedio en bonos que se está pagando. Cada una de las respuestas se lleva a cabo con los iconos función.

Finalmente, del resultado de relación que dió el proceso SUMA podemos clasificar las tuplas en orden ascendente sobre el suldo integral para obtener la relación INTEG.

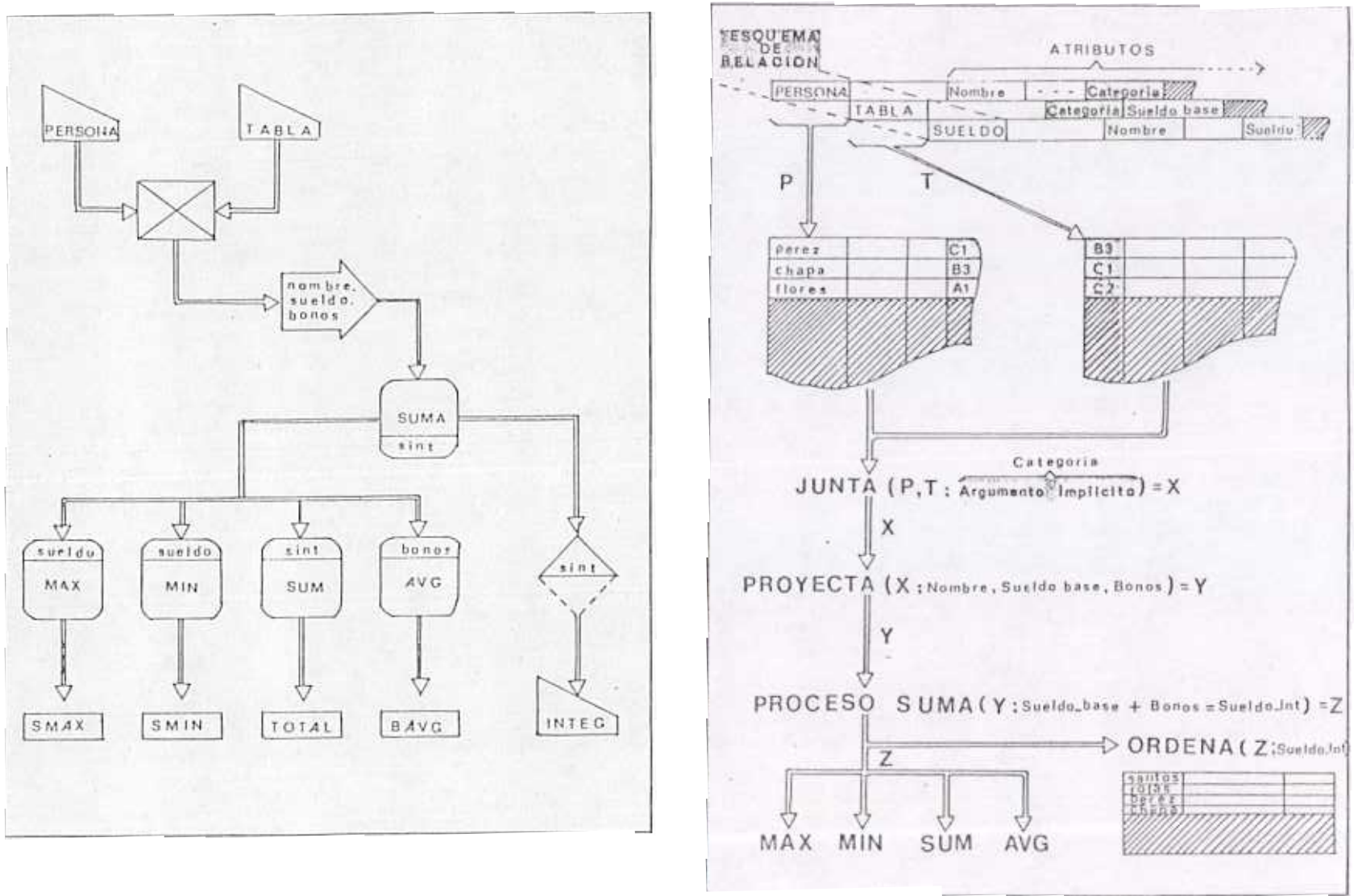


Fig 5.22 jemplo de Flujograma con su Representación de Operadores y Tablas de Relación.

5.8 CUATRO EJEMPLOS DE FLUJOGRAMAS CON SU GENERACION DE CODIGO INTERMEDIO.

EJEMPLO EJ1.

En este ejemplo la ejecución del flujograma inicia con las relaciones R1 y R2, concluyendo con el elemento terminal relación R6. Se tienen resultados parciales en las relaciones R3,R4 y R5. A partir del icono de lectura de R1 de forma paralela lo usamos para una diferencia, una junta y una selección. El constructor condicional se ejecuta bajo la condición EDAD > 25, tomando la línea de flujo verdad.

El resultado de relación R3 que fué encontrado por la diferencia de R2 con R1, debe contener el atributo SEXO para ser usado en la evaluación del constructor condicional SEXO = 'F'. La línea de flujo de datos se establece con su salida falsa.

EJ1.LIP

```
   R3 = R2 - R1
        R3 : ( SEXO = 'F'   , R4
ZZZ2! = R1 * R4
   R5 = R1 : ( EDAD > 45  )
   R6 = ZZZ2! . R5
```

EJEMPLO EJ2.

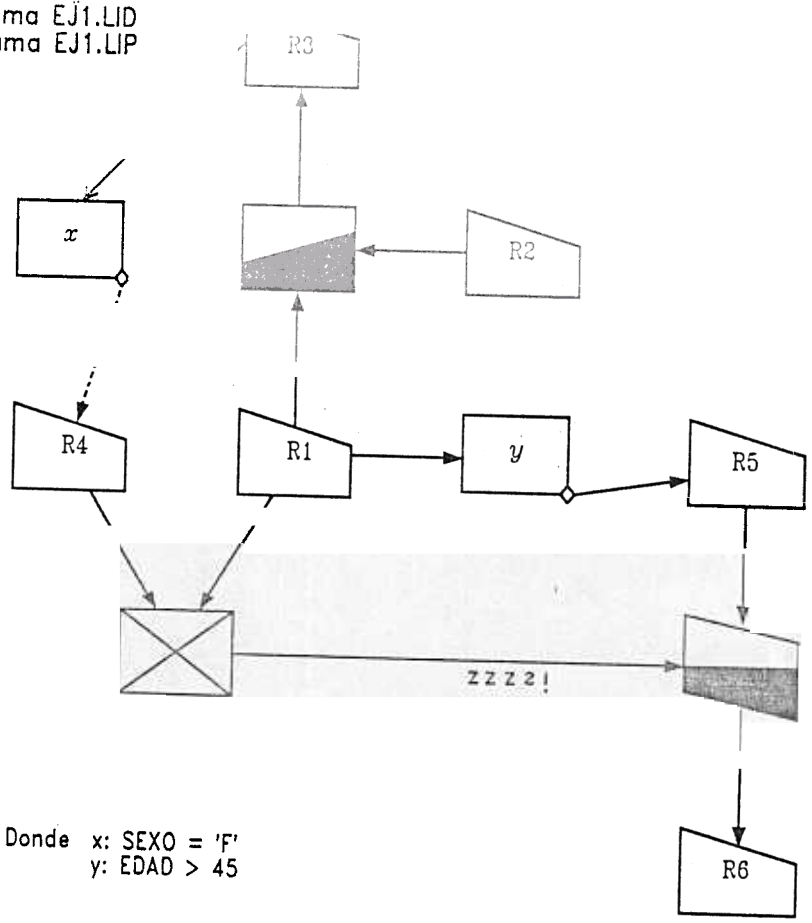
En este ejemplo partimos de la relación R1, que es usada para aplicar la función promedio (PROM) sobre el atributo EDAD, para mandar el valor al icono rectángulo y asignarle el nombre EP. Paralelamente a a esta trayectoria del flujograma, a partir de la relación R1 se tiene otro línea de flujo para llevar a cabo la operación junta con R2 como argumentos. El valor de la edad promedio (EP) es usado para un constructor condicional que toma como entrada ZZZ1! y da una salida verdadera.

Además , en este flujograma usamos el icono proceso con el nombre ACT. Este icono recibe la relación R7 para añadir un atributo sueldo con el valor 1 mill (recordando que acepta una expresión algebráica).

EJ2.LIP

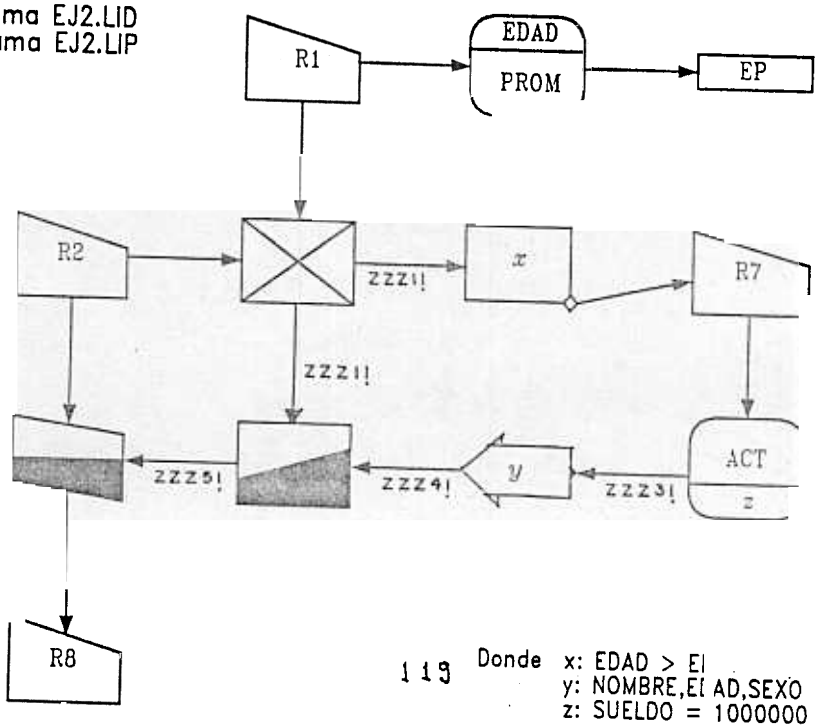
```
@EP   = PROM ( R1,EDAD ,
ZZZ1! = R1 * R2
R7    = ZZZ1! : ( EDAD > EP )
ZZZ3! = R7 [ SUELDO = 1 000 000
ZZZ4! = ZZZ3! % NOMBRE,EDAD,SEXO
ZZZ5! = ZZZ1! - ZZZ4!
R8    = R2 . ZZZ5!
```

iagrama EJ1.LID
rograma EJ1.LIP



Donde x: SEXO = 'F'
y: EDAD > 45

Diagrama EJ2.LID
Programa EJ2.LIP



Donde x: EDAD > EI
y: NOMBRE,EI AD,SEXO
z: SUELDO = 1000000

EJEMPLO EJ3

En este ejemplo iniciamos con sólo una relación R2. Lo primero es una asignación a la relación R10, para que ésta a su vez sea utilizada en una junta y una intersección. De R2 de forma paralela tenemos una salida de flujo de datos que entra a un constructor condicional que distribuye sobre dos trayectórias. La verdadera cuando la EDAD es menor que 25 y la falsa en caso contrario. Mientras la salida verdadera se asigna a R4 para ser usada posteriormente en una unión con ZZZ3!. La salida falsa se usa en una operación de junta con R10 para dar la salida ZZZ3!, que de forma paralela se usa en una diferencia con el resultado de la unión. Finalmente, se tiene que el resultado de la diferencia es ZZZ5! que se intersecta con R10 para dar el resultado en R11.

EJ3.LIP

```
R10    = R2
R4     = R2 : ( EDAD < 25 ), ZZZ1!
ZZZ3!  = R10 * ZZZ1!
ZZZ4!  = ZZZ3! + R4
ZZZ5!  = ZZZ1! - ZZZ4!
R11    = ZZZ5! . R10
```

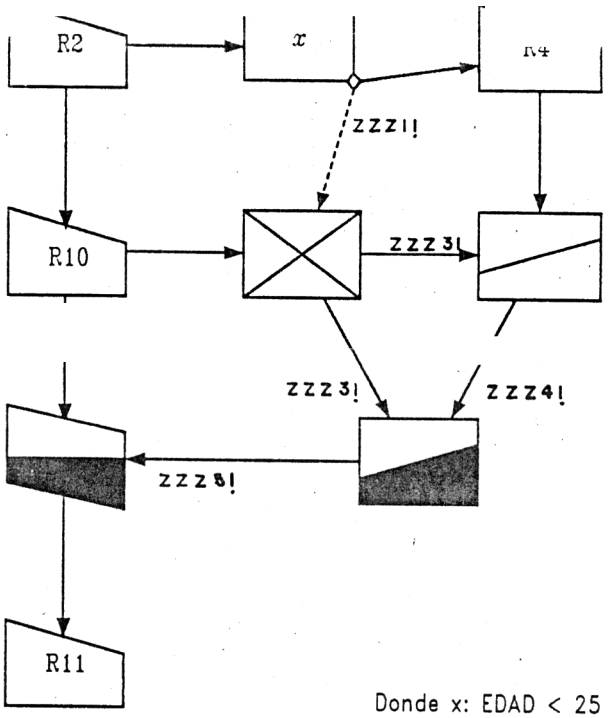
EJEMPLO EJ4.

En este ejemplo he querido ilustrar cómo puedo partir de la lectura de una relación, y desarrollar el flujograma alrededor de este icono. De R1 salen cuatro líneas de flujo de datos dos de ellas llegan a dos iconos de ordenamiento. Uno ordena la relación por EDAD en orden ascendente para dejarla en otra con el nombre R12. El segundo ordena por nombre en forma descendente para ser usado posteriormente en una unión con ZZZ2!. Además, la relación R1 es usada para encontrar por medio de la función SUM la suma delas edades, que la aloja en el icono rectángulo con nombre SE. Usamos la relación R11 para juntarla con R1 para tener el resultado ZZZ2! que se une con ZZZ1! para dar el resultado R11, que sustituye al anterior.

EJE4.LIP

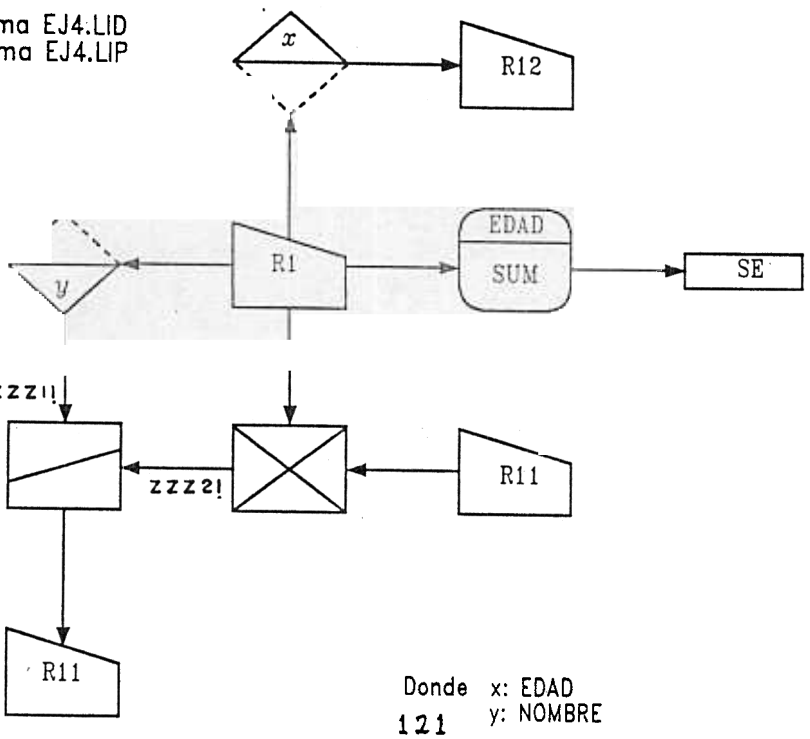
```
@SE    = SUM ( R1, EDAD
ZZZ1!  = R1 # NOMBRE,D
ZZZ2!  = R1 * R11
R11    = ZZZ1! + ZZZ2!
R12    = R1 # EDAD,A
```

Diagrama EJ3.LID
Programa EJ3.LIP



Donde x: EDAD < 25

Diagrama EJ4.LID
Programa EJ4.LIP



Donde x: EDAD
y: NOMBRE

C A P I T U L O 6

APLICACIONES Y RESULTADOS

El objetivo de este capitulo es dar una variedad de resultados del lenguaje LIDA con el fin de mostrar sus capacidades y contribuciones en estudios comparativos con casos. Se toman dos renglones principales para LIDA: uno como lenguaje de consulta en una base de datos y el otro como sistema para el desarrollo de aplicaciones.

LIDA puede usarse como lenguaje de consulta y basado en el modelo relacional de Codd, es posible la formulación de consultas con flujogramas. En este capitulo estudiamos algunos casos de consultas que son formúladas con flujogramas comparandolas con Query-By-Example y SQL. Aunque QBE y LIDA son considerados como dos lenguajes de tipo gráfico; el primero muestra las relaciones directamente visibles como objetos de datos en un esquema de tablas, manejandolos en la pantalla con el cursor a través de los renglones y columnas. En cambio, el segundo, fundamentado en un modelo de flujo de datos usa los nombres de los objetos de relación como etiquetas en las líneas de arco, para que en su turno sean instanciados y procesados en un icono. En contraste a los dos anteriores se usa el lenguaje de texto SQL para dar otro punto de vista.

Por otro lado, para ilustrar a LIDA como un lenguaje para el desarrollo de aplicaciones en este capitulo presento dos estudios de casos: decisión de crédito y facturación. El objetivo es tener resultados que sean comparados con un lenguaje de desarrollo de aplicaciones cuya naturaleza sea gráfica. Por tal razón, usamos al sistema "System for Business Automation" como una extensión de QBE, que de características gráficas, trata con tablas en dos dimensiones, formas administrativas y reportes en una pantalla de terminal. En SBA el usuario trabaja sus aplicaciones automatizando gradualmente los ejemplos, manejando los objetos de datos los dispone en un formato para que sean usados en objetos de condición y fórmulas de cálculo. Con la finalidad de mostrar una comparación entre los lenguajes gráficos, los mismos casos son resueltos con LIDA, observando que este último tiene una representación icónica más rica.

6.1 LIDA COMO LENGUAJE DE CONSULTA EN BASE DE DATOS

Un lenguaje de consulta y lenguaje de manejo de datos se definen como lenguajes de computadora de alto nivel cuyo objetivo es la recuperación, la modificación y la actualización de los datos que se encuentran en una base de datos. En un Sistema Manejador de Base de Datos (SMBD), se tienen tres componentes que permiten al usuario interaccionar con la base de datos:

1 El Lenguaje de Definición de Datos (LDD), que permite definir esquemas y subesquemas.
2) El Lenguaje de Manejo de Datos (LMD), usado para modificar y actualizar datos; y finalmente,
3) El Lenguaje de Consulta (LC) que permite resolver problemas de preguntas a la base de datos.

En muchos de los casos de SMBD la implantación de los diferentes lenguajes se combinan bajo el mismo punto de vista del modelo de datos, usando una arquitectura constituida fundamentalmente de cuatro componentes: a) procesador de interfaz con el usuario, b) procesador de esquemas, c) ayudas en la administración de la base de datos, y c) gestor de la base de datos.

El procesador de interfaz de usuario incluye traductores del LC, lenguajes de programación anfitriones y traductores del LMD; así como, herramienta funcional que ofrece facilidades de comunicación del usuario con la máquina, basadas en gráficas y lenguaje natural. El desarrollo de los lenguajes de consulta y manipulación de datos en general han seguido por dos corrientes. La primera corresponde más a la fundamentación como lenguajes de programación y la teoría de base de datos; mientras que la segunda plantea la elaboración de interfaces determinada por el concepto de funcionalidad con interacciones más simples para el usuario.

El primer grupo se concentra en los desarrollos basados en formas sintácticas y semánticas con la interacción a la base de datos. Los lenguajes desarrollados en este contexto se caracterizan por la especificación completa de una operación dada por un comando o una secuencia de ellos. Se fundamentan con lenguajes formales cuyas orientaciones son de tipo matemático, extendiendose a lenguajes naturales que usan vocablos claves del idioma. El objetivo es desarrollar interfaces en lenguaje natural restringido, para hacer mas amigables los sistemas manteniendo la capacidad funcional del sistema.

El segundo grupo de lenguajes tiene su énfasis de desarrollo en consideraciones de tipo ergonómicas asociadas a la interacción del usuario con el equipo de cómputo. En este grupo se incluyen sistemas simples que usan teclas como funciones, selección de menus e interacciones gráficas con las bases de datos. Este grupo de desarrollo representa una corriente hacia mayores capacidades de funcionalidad.

La linea divisoria entre los dos grupos es aparente, la tendencia es integrar los dos aspectos de funcionalidad y de formalismo, aplicando sucesivamente transformaciones automáticas a lenguajes intermedios asociados a un sistema matemático formal o un modelo de datos.

En la teoria de base de datos se destaca la importancia de los lenguajes relacionales, como uno de los principales que han sido implementados cada vez más en SMBD. En el modelo relacional de Codd -usado en este trabajo- se consideran tres sistemas formales sobre los cuales se pueden desarrollar lenguajes de consulta. El sistema algebráico en el cual se basa el sistema "Information System Base Language" (ISBL), cálculo de dominios para "Query-By_Example" (QBE) y finalmente cálculo de tuplas asociado a "Structured Query Language" (SQL).

El primer caso es un sistema que se basa fundamentalmente en el uso de operadores relacionales de alto orden con una sintaxis en términos de expresiones algebráicas. En el segundo sistema las variables representan valores sencillos en los dominios de atributos mostrandose excelente al lenguaje QBE. En Query-By-Example se tiene un esquema de tablas con una representación gráfica en donde se especifican las variables sobre el dominio de los atributos. Por último, el cálculo tuplas es un sistema formal que maneja los operadores relacionales en una notación de teoria de conjuntos. Uno de los lenguajes más representativos basado en el sistema formal de tuplas es SQL; con estructura de programación basada en bloques y el uso de palabras claves en el idioma inglés.

Las líneas de investigación actuales en lenguajes de consulta a base de datos, tienen como objetivo elevar el nivel de abstracción con tendencias a lenguaje natural e interfaces entre la máquina y el usuario, que brinde facilidades de tipo gráfico e incorporación de factores ergonómicos. El uso de toda esta herramienta de "software" por parte del usuario final conlleva a la necesidad de integrar módulos de proceso de interfaz que dejan lenguajes anfitriones e intermedios, los cuales suelen ser algunos de los mencionados.

6.1.1 SQL: Structured Query Language

La primera version de SQL fué desarrollada en 1974 por Chamberlain [CHAM74], tomando como base al lenguaje SQUARE que tiene énfasis en una sintáxis matemática. SQL es un lenguaje de consulta estructurado cuya transformación de bloques SELECT-FROM-WHERE, es la parte principal de una representación sintactica del lenguaje.

```
SELECT A1, A2, A3, ...,An
FROM   R
WHERE  B1 O1 b1 and ... and Bm Om bm
```

En donde Ai son atributos, R es el nombre de la relación y los Bi Oi bi son fórmulas de relación en donde el operador Oi puede tomar los simbolos de relación =, >, <, =>, =< y not. Las caracteristicas principales del lenguaje SQL se presentan de forma breve en seguida.

Recuperación Simple.- Esta transformación se considera como una formación de un subconjunto horizontal seguida por una proyección; en donde la proyección opera extrayendo columnas especificadas y luego eliminando los renglones redundantes de las columnas extraidas. Sin embargo, en SQL, en general no se elimina el duplicado de los resultados a menos que el usuario los solicite de modo explicito por medio de la palabra clave UNIQUE.

Recuperación Calificada.- En este tipo de recuperación el predicado que sigue a WHERE puede incluir los operadores de comparación asi como operadores booleanos.

Recuperación con Ordenamiento.- En este tipo de recuperación el usuario especifica que el resultado se ordene de una manera particular antes de ser desplegado.

Recuperación de más de una Tabla.- Aquí el usuario puede dar el nombre de varias tablas en la cláusula FROM y puede usar los nombres de las tablas como cuantificadores en la cláusula SELECT y WHERE, es por así decirlo una forma de reunión de información.

Recuperación usando ANY.- Este tipo de recuperación realiza una subconsulta de una consulta. De este tipo de recuperación se deriva otra que es menor que ANY, entendiendose por esto que un cierto predicado es menor que un valor actual de ese predicado.

Recuperación usando IN.- Este tipo de recuperación puede equipararse con el operador pertenece a un conjunto dado y es equivalente a la recuperación =ANY.

Recuperación con Niveles Múltiples de Anidamiento.- Este tipo de recuperación solamente se utiliza para anidar un número cualquiera de subconsultas a cualquier profundidad.

Recuperación usando ALL.- Este tipo de recuperación toma un valor booleano si toda la transformación cumple con una determinada condición.

Recuperación usando EXISTS.- En este tipo de recuperación la clausula Exists representa el cuantificador existencial, y es utilizado para saber si existe información en alguna tabla o archivo.

Recuperación utilizando NOT EXISTS.- La manera de uso de esta cláusula es la de descomponerla en consultas más pequenas.

Recuperación usando UNION.- Este tipo de recuperación es similar al operador Union de teoria de conjuntos.

Recuperación usando a NULL.- cuando un valor nulo se compara con algún otro valor al evaluar un predicado, cualquiera que sea el operador de comparación del resultado nunca es verdadero.

Funciones en SQL.- En problemas prácticos las consultas involucran funciones típicas como son:

COUNT: Cuenta el número de valores o tuplas.

SUM: Suma de valores sobre un dominio.

Promedio de los valores sobre un dominio.

MAX: El valor máximo de los valores de un dominio

MIN: El valor mínimo de los valores de un dominio

Operador Agrupa Por (GROUP BY).- El operador agrupa permite en SQL recuperar un lote de tuplas que tienen un mismo valor especificado por el atributo especificado después del operador (Group by A1). Este operador permite particionar la tabla en grupos que tiene el mismo valor sobre un dominio. Este operador se asocia a una función que se evalua sobre otro campo y permita obtener el valor a cada partición.

6.1.2 QBE: QUERY-BY-EXAMPLE.

El nombre del lenguaje Query-by-Example se usa para referirse al mismo sistema de consulta que se basa en el cálculo de dominios del modelo relacional. En contraste al lenguaje SQL que es un lenguaje de texto que usa palabras claves, en QBE el usuario efectúa las solicitudes con ejemplos, que construye mediante el desplegado de objetos dentro de una forma de esquema de tablas presentado en la pantalla.

El lenguaje tiene una sintáxis bidimensional que permite formular las consultas insertando un ejemplo de una respuesta moviendo el cursor libremente a lo largo de los renglones y las columnas de las tablas. La forma gráfica tanto de las consultas como del despliegue son esqueletos de tablas.

RELACION

A1	A2	A3	A4	
P.XXXX			PARIS	

La clave "P." determina un operador para imprimir todos los valores del dominio del atributo por medio del elemento ejemplo que esta subrayado. El valor Paris determina una recuperación calificada en donde la consulta solo recupera las tuplas que complen con el valor en el atributo. En QBE existe una distinción entre un elemento constante y un elemento ejemplo, el último que es variable siempre se subraya.

Recuperación Simple.- Esta recuperación actúa de la misma forma que en SQL, con la diferencia que QBE elimina automáticamente los valores redundantes. Para suprimir esa eliminación el usuario puede especificar la palabra clave ALL.

Recuperación Calificada.- En esta recuperación podemos usar todos los operadores de comparación (<, >, =<, =>, ¬, =). En el caso del último operador de igualdad solo con poner el elemento constante se toma implícito. En el caso de recuperación calificada tenemos la posibilidad de juntar dos condiciones con AND y OR. La primera se puede especificar en un solo renglón, mientras el OR requiere especificarse en dos renglones por separados.

Recuperación con Ordenamiento.- En este tipo de recuperación la información recuperada es ordenada en forma descendente utilizando las cláusulas DO. y AO. para un orden ascendente. Este tipo de recuperación permite un tipo de ordenamiento usando varios campos, primario y secundario con << AO (1) >> y << AO (2) >>.

Recuperación Usando Enlaces.- En este tipo de consultas se despliegan varias tablas generando estructuras vacias y llenando los espacios correspondientes al encabezado y los campos. En este caso el mismo elemento ejemplo es usado en ambas tablas sobre el atributo deseado. Este caso es típico en la aplicación de la junta.

Recuperación Usando Caja de Condiciones.- En QBE se tiene de forma adicional la caja de condiciones cuyo encabezado es CONDITIONS. Esta caja puede ser desplegada por el usuario con la finalidad de insertar una condición por separado. Esta caja se obtiene usando otra llave de función en la terminal. Las diferentes expresiones se introducen por separado en una caja, pero QBE las considera simultaneamente.

Funciones Integradas.- QBE también considera un conjunto de funciones integradas:

CNT.ALL.	CNT.UNQ.ALL.	
SUM.ALL.	SUM.UNQ.ALL.	ELIMINA REDUNDANCIAS
AVG.ALL.	AVG.UNQ.ALL.	
MAX.ALL.		
MIN.ALL.		

6.1.3 ESTUDIO DE CASOS

En seguida vamos a dar inicio al estudio de algunos casos cuyo desarrollo se presenta a partir del planteamiento del problema. Se tiene una base de datos la cual esta constituida de tres esquemas de relación: PROVEEDOR, PARTE y ORDEN:

ESQUEMA DE RELACION PROVEEDOR:
PROVE (#S, SNOMB, EDO, CIUDAD

Relación PROVE: TABLA 1.

ESQUEMA DE RELACION PARTE:
PARTE (#P, PNOMB, COLOR, PESO, CIUDAD

Relación PARTE: TABLA 2.

ESQUEMA DE RELACION ORDEN:
ORDEN (#S, #P, CANTIDAD)

Relación ORDEN: Tabla 3.

TABLA 1. RELACION PROVEEDOR.

PROV

#S	SNOMB	EDO	CIUDAD
S1	perez	20	londres
S2	flores	10	paris
S3	chapa	30	paris
S4	rojas	20	londres
S5	reyes	30	atenas

TABLA 2. RELACION PARTE

PARTE

#P	PNOMB	COLOR	PESO	CIUDAD
P1	desarmador	rojo	12	londres
P2	pinzas	verde	17	paris
P3	llaves	azul	17	roma
P4	alicates	rojo	14	londres
P5	pijas	azul	12	paris
P6	palas	rojo	19	londres

TABLA 3. RELACION ORDEN.

ORDEN

#S	#P	CANTIDAD
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S4	P2	300
S4	P4	300
S4	P5	400

CONSULTA Q1.

LN) Obtenga el nombre del proveedor y su número para los proveedores que se encuentran en la ciudad de Paris y tengan en edo. mayor que 20.
Listalos en forma descendente por EDO.

```
SQL) SELECT   SNOMB, #S
     FROM     PROVE
     WHERE    CIUDAD = 'paris'
     AND      EDO > 20
     ORDER BY EDO DESC
```

QBE)

PROVE

#S	SNOMB	EDO	CIUDAD
P.SN	P.SX P.SX P.SZ	> 20 P.DO.ST	paris

Fig. 6.1 Consulta Gráfica de Q1 en QBE.

CONSULTA Q2

LN) Obtenga los nombres de aquellos proveedores que suministran la parte P2.

SQL)

```
SELECT SNOMB
FROM   PROVE
WHERE  # S = ANY ( SELECT #S
                   FROM   ORDEN
                   WHERE  #P = "P2" ).

SELECT UNIQUE SNOMB
FROM   PROVE, ORDEN
WHERE  PROVE. #S = ORDEN. #S
       AND  ORDEN.#P = "P2".

SELECT SNOMB
FROM   PROVE
WHERE  #S IN
            ( SELECT #S
              FROM   ORDEN
              WHERE  #P = "P2" )
```

QB)

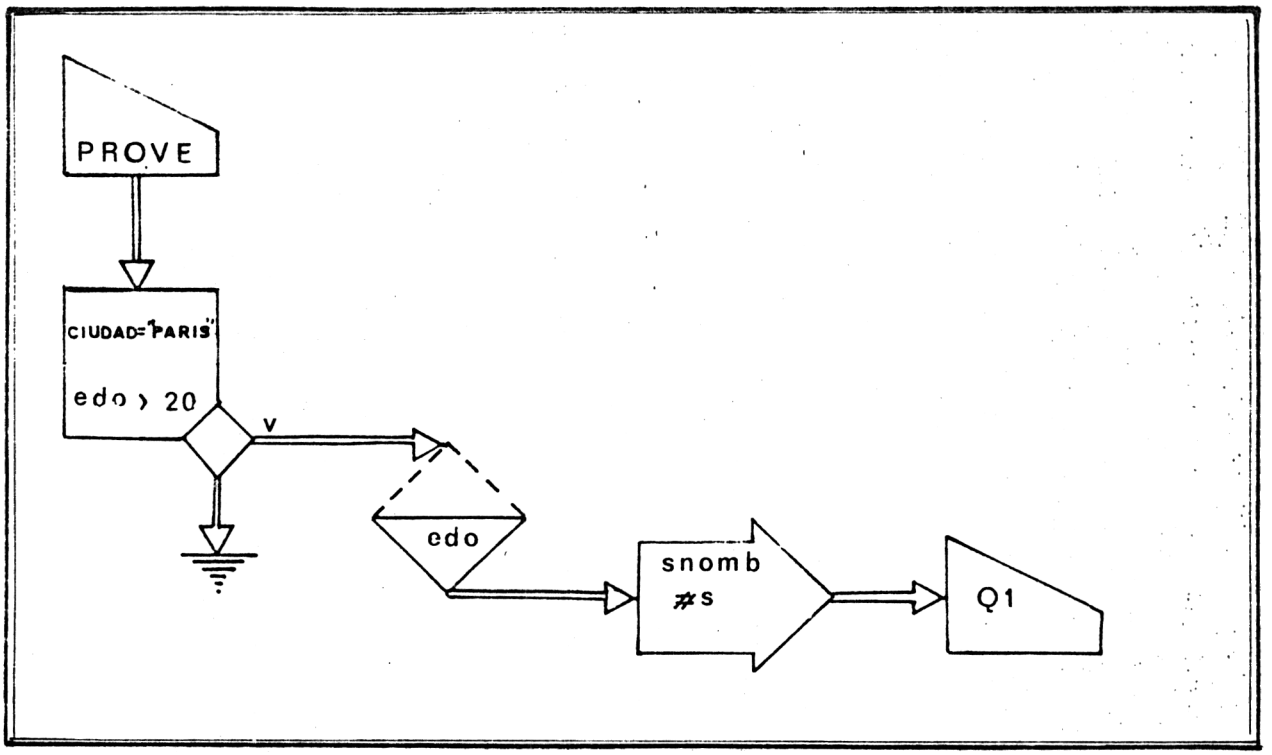
PROVE

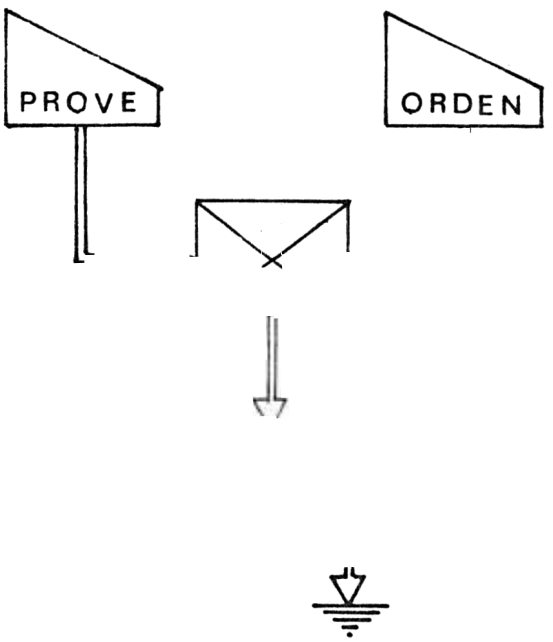
#S	SNOMB	EDO	CIUDAD
SX	P.SX		

ORDEN

#S	#P	CANTIDAD
SX	P2	

Fig. 6.2 Consulta Gráfica de Q2 en QBE


PROVE
CIUDAD='PARIS'
edo > 20
v
edo
snomb
#s
Q1


PROVE
ORDEN

CONSULTA Q3

LN Obtenga los nombres de los proveedores los cuales uminis-
tran al menos una parte de color rojo.

```
SELECT SNOMB
FROM   PROVE
WHERE  #S  IN
           ( SELECT #S
             FROM   ORDEN
             WHERE  #P     IN
                             SELECT #P
                             FROM   PARTE
                             WHERE  COLOR = "ROJO"
```

PROVE

#S	SNOMB	EDO	CIUDAD
SX	P. SN		

ORDEN

#S	#P	CANTIDAD
SX	PX	

PARTE

#P	PNOMB	COLOR	PESO	CIUDAD
PX		ROJO		

Fig. 6.5 Consulta Gráfica de Q3 en QBE

RECUPERACION USANDO EXIST Y NO EXIST.

CONSULTA

LN

Obtenga los nombres de los proveedores los cuales suministran la parte P2.

SQL)

```
SELECT SNOMB
FROM   PROVE
WHERE  EXIST

       ( SELECT *
         FROM   ORDEN
         WHERE  #S = S. #S
         AND    #P = 'P2'
```

CONSULTA Q4.2

LN

Obtenga los nombres de los proveedores para los cuales no se suministra la parte "P2".

SQL)

```
SELECT SNOMB
FROM   PROVE
WHERE  NOT EXIST
                    SELECT *
                    FROM   ORDEN
                    WHERE  #S = S.#S
                    AND    #P = "P2" )
```

QB

PROVE

#S	SNOMB	EDO	CIUDAD
SX	P.SN		

ORDEN

#S	#P	CANTIDAD
SX	P2	

Fig. 6.6 Consulta Gráfica de la Consulta Q4.1 en QBE.

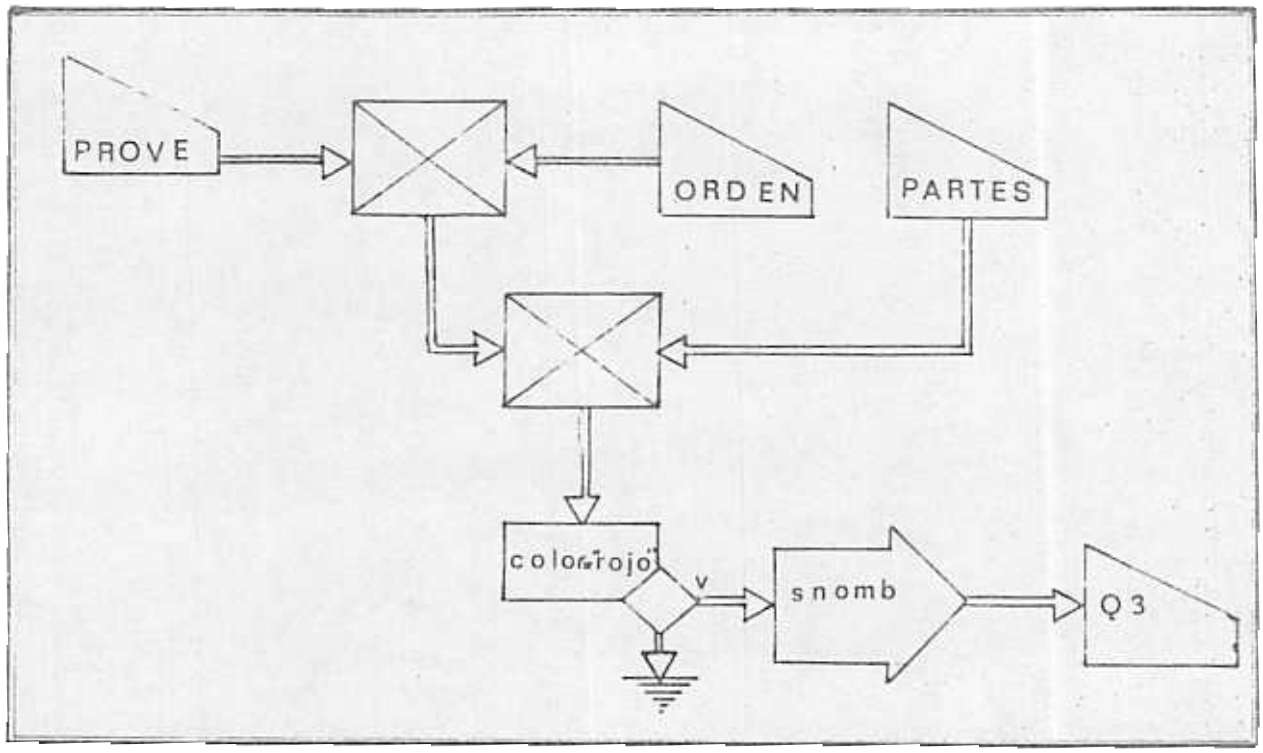


Fig. 6.7 CONSULTA Q3 EN FLUJOGRAMA

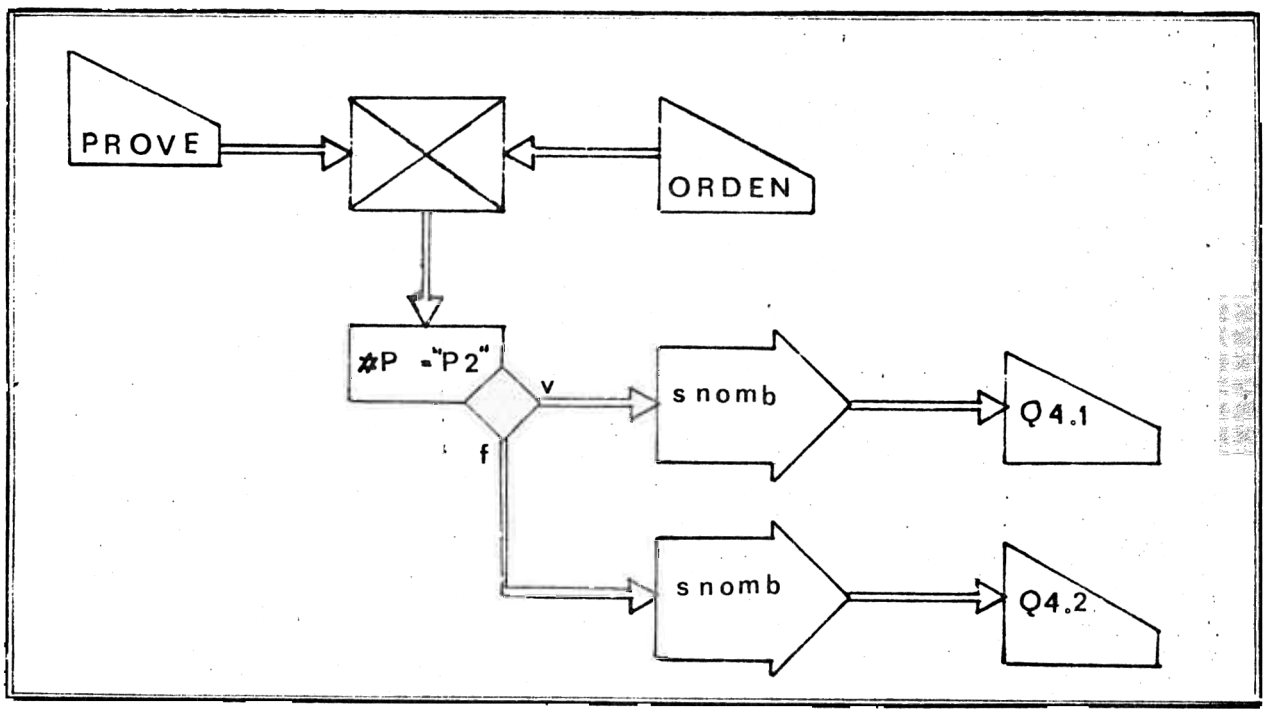


Fig. 6.8 CONSULTA Q4. Y Q4.2 EN FLUJOGRAMA

RECUPERACION USANDO UNION.

CONSULTA Q5

LN)

Obtenga el número de las partes cuyo peso es mayor que 18 Kgs., o bien son suministrados por el proveedor S2.

SQL)

```
SELECT #P
FROM   PARTE
WHERE  PESO > 18

UNION

SELECT #P
FROM   ORDEN
WHERE  #S = 'S2'
```

RECUPERACION USANDO FUNCIONES.

CONSULTA Q6

Obtenga el número de proveedor para los que suministran con estado menor que el máximo corriente.

```
SELECT #S
FROM    PROVE
WHERE   EDO <
              ( SELECT MAX (EDO)
                FROM   PROVE

SELECT #S
FROM   PROVE
WHERE  EDO < ANY ( SELECT EDO
                   FROM   PROVE
```

QB)

PROVE

#S	SNOMB	EDO	CIUDAD
P.SX		ST P.MAX. ALL. SS	

CONDICION
ST < MAX. ALL. SS

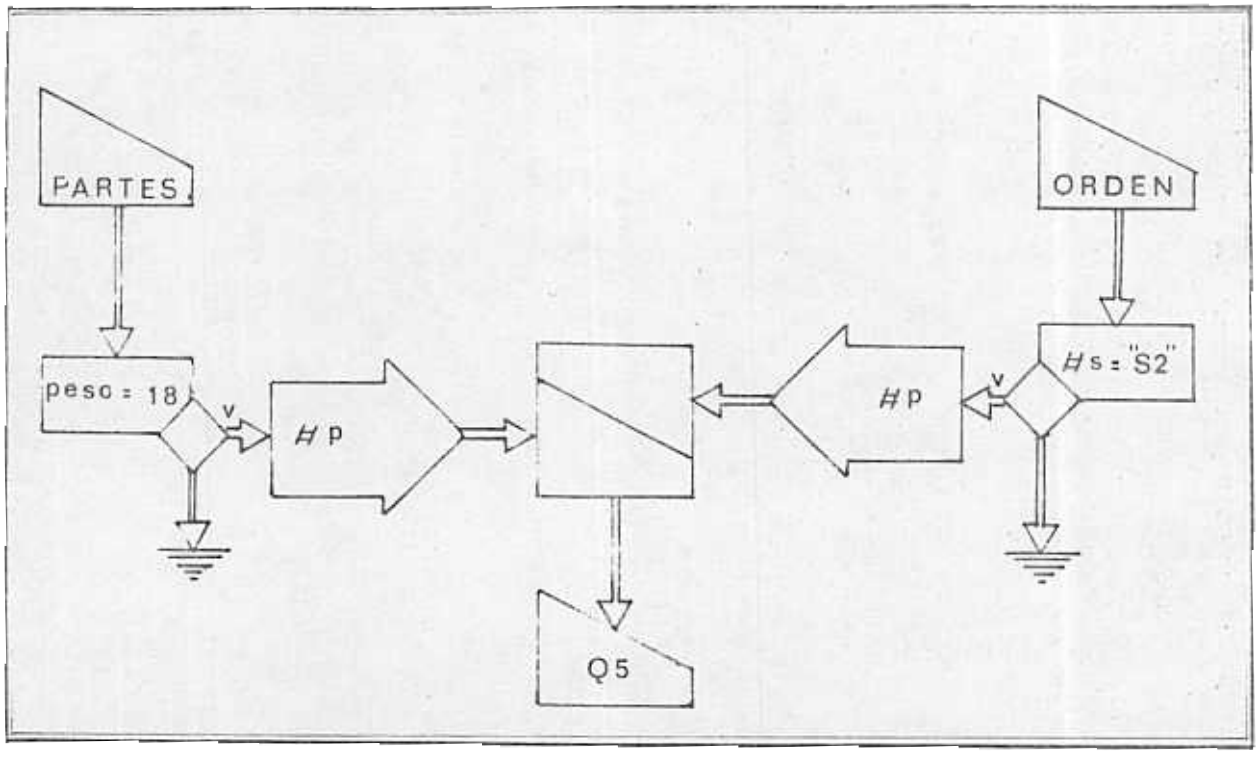
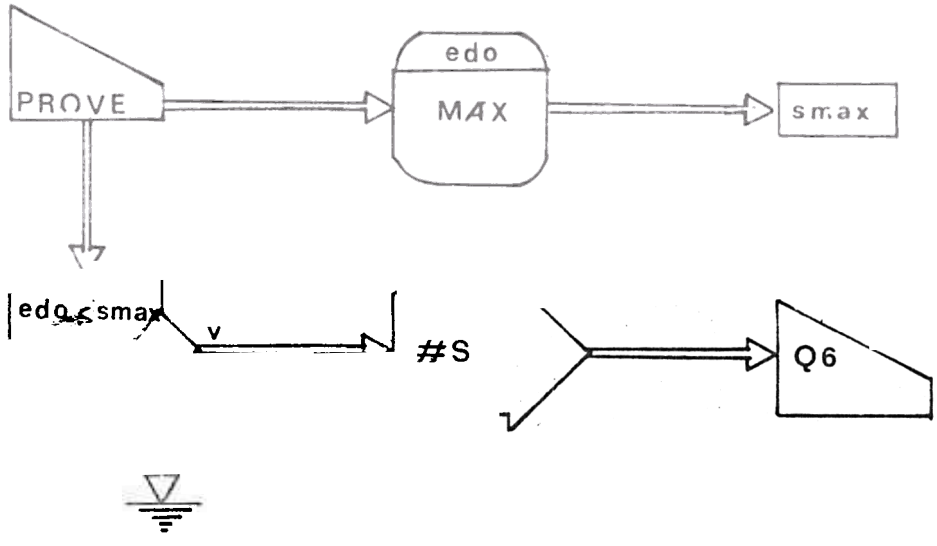


Fig. 6.11 CONSULTA Q5 EN FLUJOGRAMA.



CONSULTA Q6 EN FLUJOGRAMA

2 LENGUAJE LIDA PARA EL DESARROLLO DE AP ICACIONES

El objetivo de esta sección consiste en resolver dos casos de desarrollo de aplicaciones con el lenguaje SBA como lenguaje gráfico y LIDA en su mismo caracter. La finalidad es un estudio comparativo del desarrollo de los programas en SBA vs. flujogramas en LIDA, de un problema de procesamiento de ordenes en decisiones de crédito y otro de facturación.

SBA es un sistema para desarrollar aplicaciones mediante el manejo de la información en gráficas de tablas en dos dimensiones, formas administrativas y reportes en pantalla de terminal. El usuario puede automatizar gradualmente su aplicación por medio de "ejemplos" como sucede en QBE.

El sistema esta dividido en tres subsistemas bajo la misma filosofía del lenguaje: el lenguaje QBE para base de datos; lenguaje de programación para la automatización de las aplicaciones; y un lenguaje para la especificación de sistemas administrativos. Un programa SBA está compuesto de una colección de transacciones QBE sobre las tablas o mas objetos de datos complejos tales como formas y reportes. El programa usa ejemplos por especificación de invocación de programas, parámetros, gatillos e interacción de usuarios.

6.2.1. Estudio del Caso: DECISION DE CREDITO

El siguiente caso muestra como un gerente de crédito automatiza su trabajo con la descripción del lenguaje de programación SBA, con un estudio comparativo de solución en flujogramas.

El programa presenta el trabajo del gerente de crédito en términos de una forma de ORDEN que el recibe para tomar decisiones de crédito, presentandolo en una forma de DECISION DE CREDITO. El primer proceso consiste en automatizar la función clerical de copiar el número de orden de la forma ORDEN a una forma de salida DECISION DE CREDITO; determinado fundamentalmente como una invocación de consulta tipo QBE.

De este programa el gerente de crédito dá un ejemplo de quizas al atributo DECISION para que sea evaluado que SI, de acuerdo al cumplimiento de la razón de crédito igual a A1, que se determina en el atributo de clientes con el ejemplo TIME en el archivo de crédito. Además, se incluye la decisión de crédito de SI dependiendo del complimiento de la restricción de una cantidad menor de 10,000 solicitada directamente en la orden; figura 6.13 (a) para el caso de SBA y figura 6.13 (b) para el caso de flujograma desarrollado en LIDA.

En una segunda parte, el programa rechaza la orden por colocar un NO en la columna de decisión en la forma de CREDITO DE DECISION cuando los clientes tienen una capacidad de crédito menor o igual B2, presentada en la figura 6.14 (a) para el sistema SBA y la figura 6.14 (b) para LIDA respectivamente.

PROGRAM	NOMBRE
	CREDITO OK

ENTRADA

ORDEN		
numero	cliente	cantidad

SALIDA

DECISION CRED	
# orden	decision

ORDEN		
numero	cliente	cantidad
5	time	$100

DECISION CRED	
# orden	decision
5	maybe

CREDITO	
cliente	tasa
time	record

CONDICION	CALCULO
[record = A1 $100 < 10k]	maybe = SI

Fig. 6.13 (a) PROGROMA EN SBA PARA DECISION DE CREDITO (Fase I)

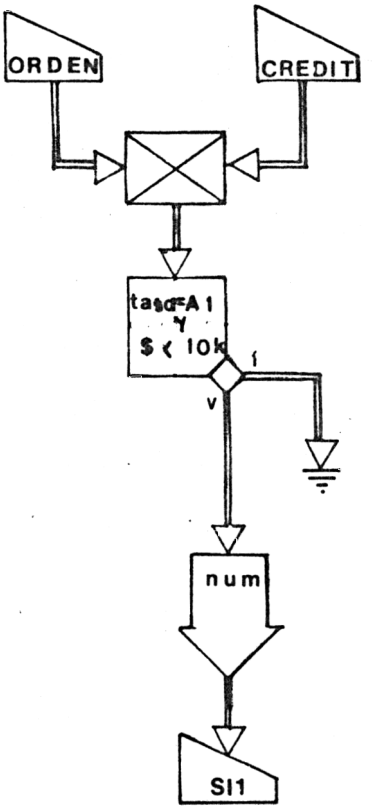


Fig. 6.13(b) FLUJOGRAMA EN LIDA
PARA DECISION DE CREDITO (Fase I)

| DECISION

maybe

| ORDEN
| numero | cliente
5 time

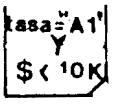
5

| CREDITO
| cliente
| time

| CONDICION | CALCULO

record =
$100 <

ELSE | record =< B2 | maybe = no



Finalmente, el programa deberá calcular una decisión de crédito debido a la capacidad de endeudamiento. Esta se determian determinada por medio de A2 y el valor del balance menor que 100,000.

El cálculo del balance se obtiene con el crédito corriente dado en el libro de ventas más los pendientes correspondientes al de diario. Lo anterior resuelve, por último, el problema a la decisión de crédito. En el caso de la solución por flujograma en la figura 6.16, los archivos de salida SI1 y SI2 son relaciones unarias de número de cliente, que consideran a quien se le dá crédito y los que no.

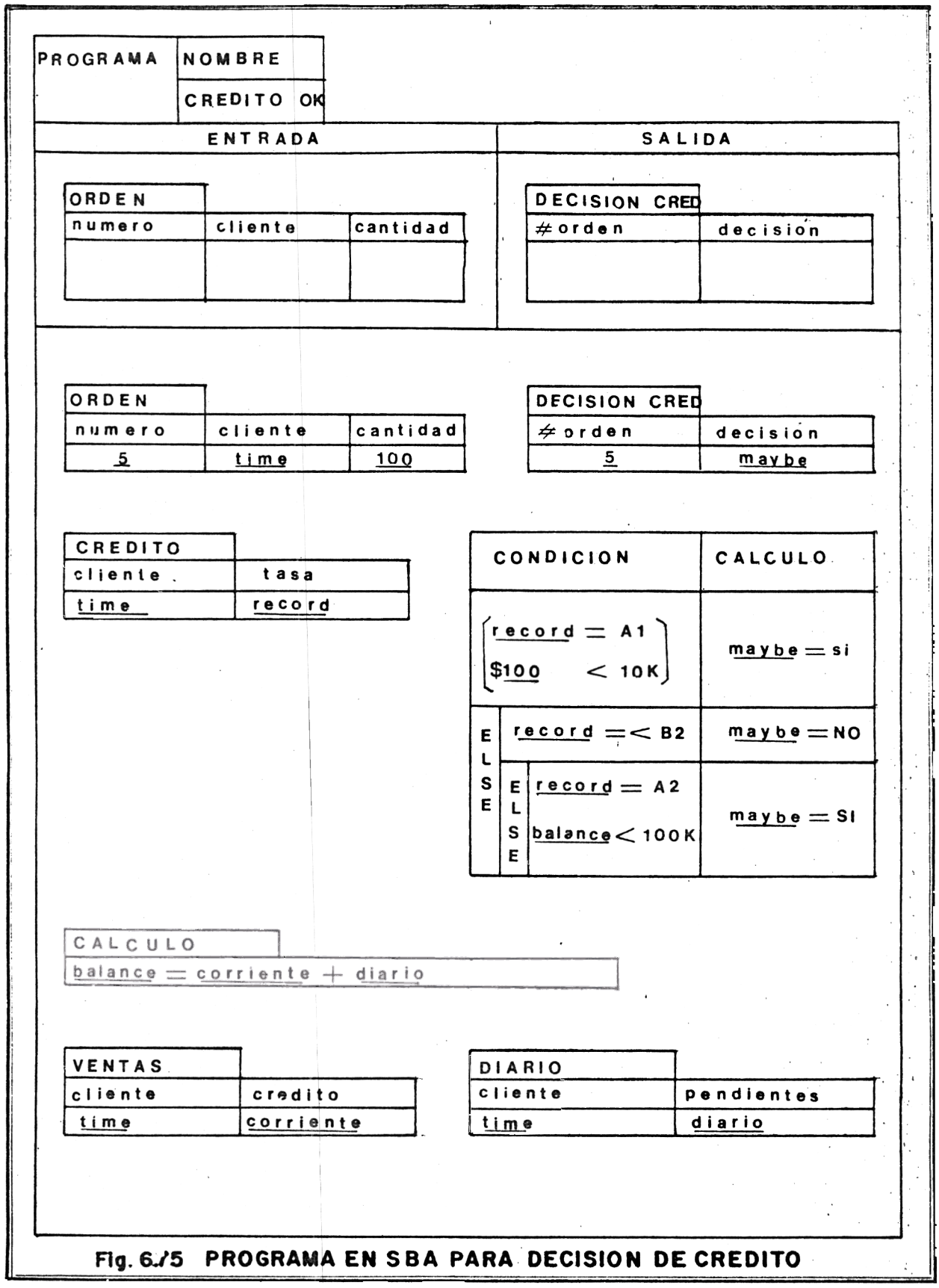


Fig. 6.15 PROGRAMA EN SBA PARA DECISION DE CREDITO

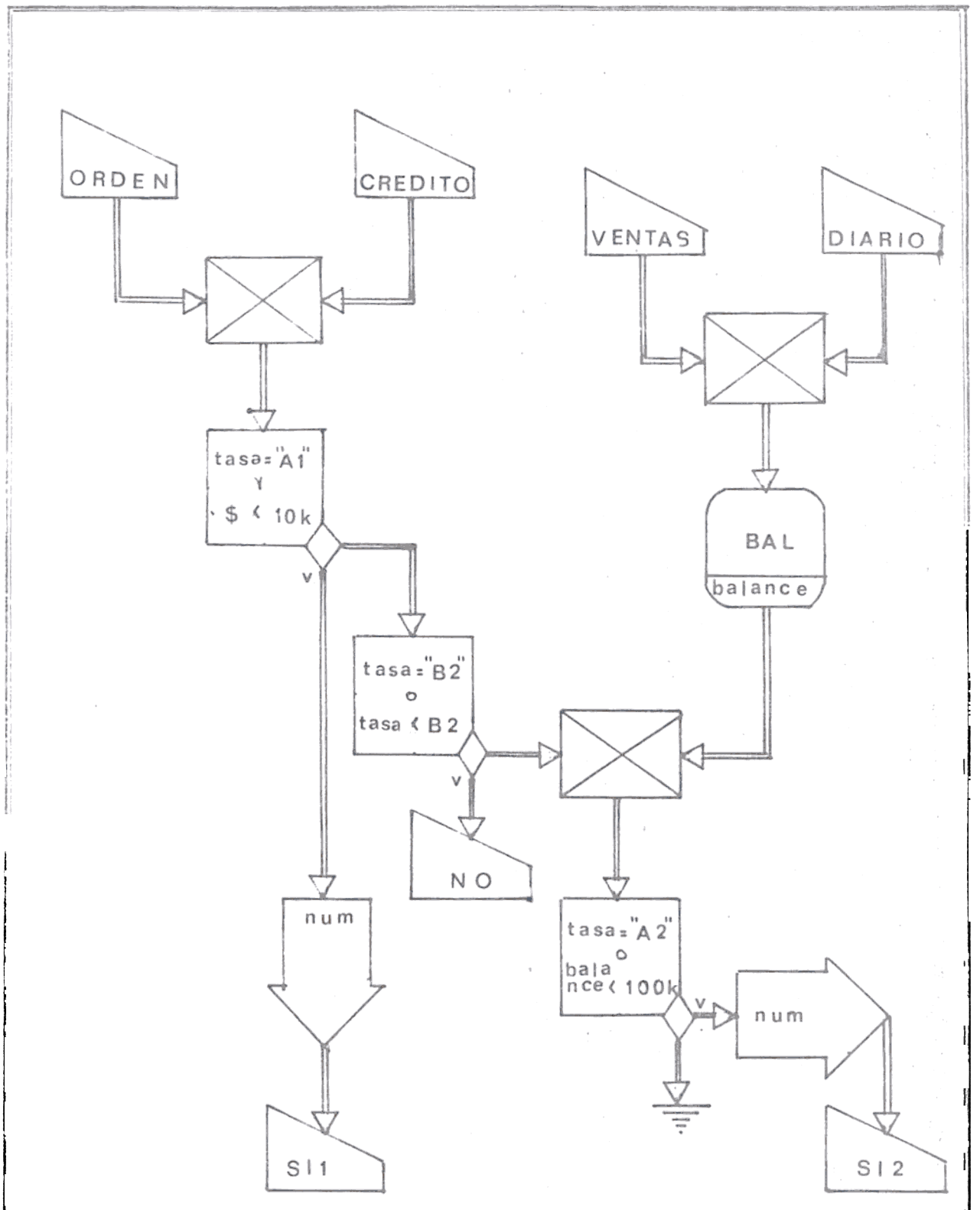


Fig. 6.16. FLUJOGRAMA EN LIDA PARA DECISION DE CREDITO

6.2.2 Estudio del Caso: FACTURACION.

Un ejemplo tipico de programas de aplicación es el de facturación, el cual recibe ordenes como entradas y produce facturas como salida, tomando en cuenta que se tienen muchas ordenes por cliente.

El programa descrito en la figura 6.17 en el lenguaje SBA especifica una entrada completa de ordenes para ser procesadas por el programa.

En el programa figura 6.17 el ejemplo ALL.ABC en la forma de orden, especifica que lote de ordenes son procesados de una vez para cada cliente. El ejemplo ABC en la facturación especifica que las facturas son llenadas para cada cliente. La SUM.ALL 15 en ordenes, suma todas las cantidades para todas las ordenes, para cada cliente y para cada articulo. La suma por articulo es colocada en la factura.

El costo del articulo I.COST es calculado por observar el precio en el archivo maestro PRECIO, para ser colocado en la columna I.COST de la factura para cada producto, calculandose el monto con la asignación del ejemplo 61.

La cantidad total para todos los articulos se calcula y coloca en el campo de TOTAL. Y el descuento para los clientes se considera con el ejemplo 3%, que efectúa el calculo de descuento de acuerdo a la clase A, B y C.

El programa presenta un refinamiento, considerando que además, la clase B tiene como ejemplo 5% para ser usada a traves de una condición adicional. En lugar del 5% de descuento, el descuento se lleva a cabo de acuerdo a la razón de crédito de los clientes, con el cambio de constante 5% al cambio de ejemplo 5%. Dicho descuento se efectua añadiendo una nueva tabla de registro de los clientes con condiciones 1, 2 y 3 que determinan un descuento del cuatro, cinco y seis por ciento respectivamente.

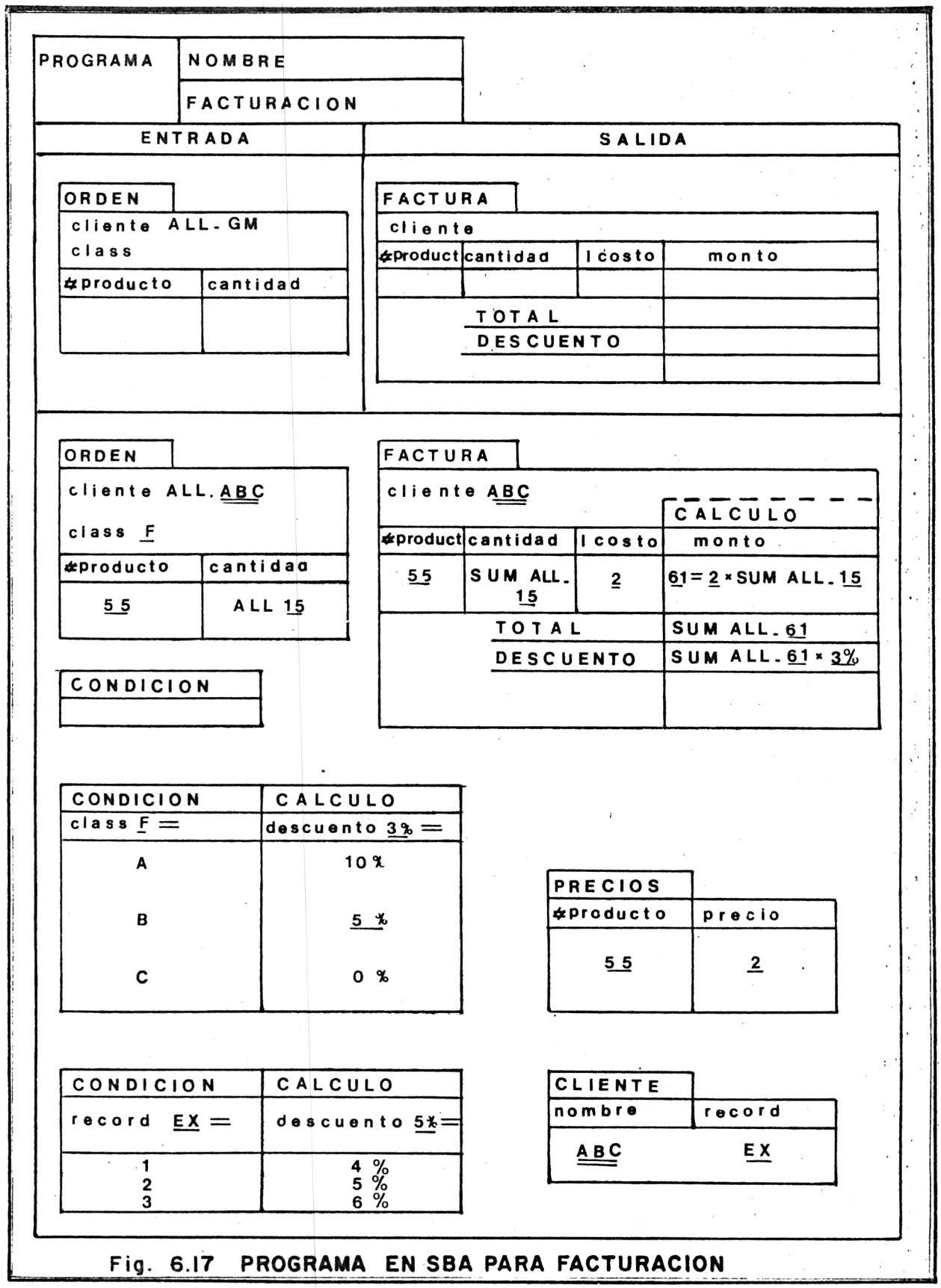


Fig. 6.17 PROGRAMA EN SBA PARA FACTURACION

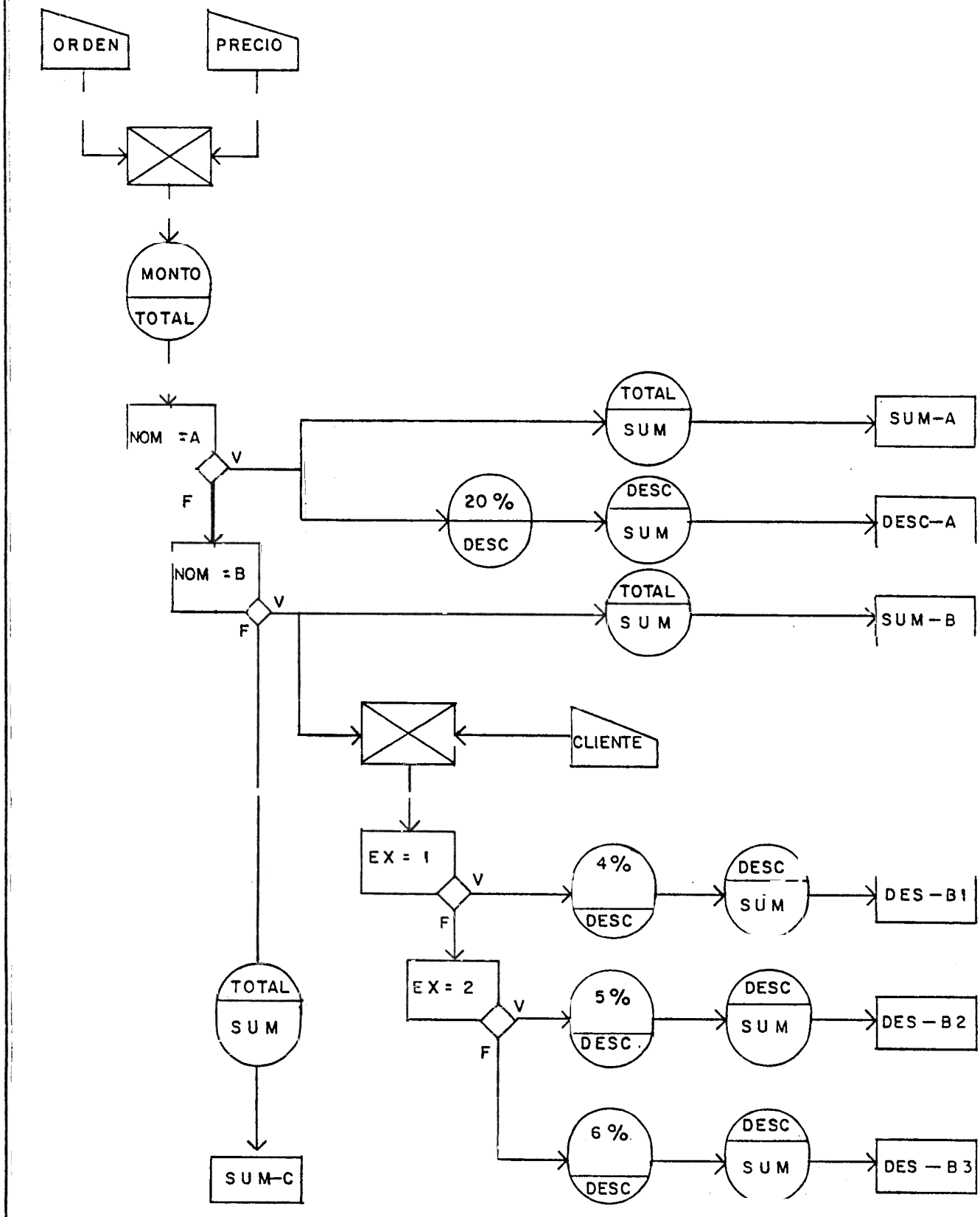


Fig 8 FLUJOGRAMA EN LIDA PARA FACTURACION

C A P I T U L O

DESCRIPCION DEL SISTEMA LIDA

7.1 Estrucrura Interna de los Flujogramas

Un flujograma es tratado internamente como una matriz de apuntadores, en donde cada uno aloja un nodo. Estos son elementos del flujograma y corresponden a iconos con sus argumentos y parámetros. Cada que se genera un elemento en el diagrama, se crea dinámicamente en memoria. La estructura de datos asociada a cada elemento contiene los campos para almacenar la información que determinan las características visuales, comportamiento dentro del flujograma y las ligas con los demás elementos.

Iniciamos con la estructura básica de matríz de dibujo. La matríz de apuntadores DIBUJO aloja cada uno de los nodos que se van generando. Cuando se inserta un nuevo elemento al flujograma, el apuntador correspondiente a su posición cartesiana es orientado hacia la dirección de memoria donde se reserva el espacio necesario para almacenar los campos correspondientes a su estructura de datos; dada por la siguiente definición:

```
DIBUJO : array  1. . MAXR , 1. .MAXC  of ^ ELEMENTO;
```

donde ELEMENTO es el nombre de la estructura de datos de un nodo, com MAXR y MAXC como números de renglones y columnas que componen la matríz correspondiente.

Las estructuras que determinan las características visuales y funcionales de cada icono del lenguaje LIDA, se generan automáticamente residiendo en memoria durante el periodo de trabajo. Cuando se inserta un nuevo elemento en el flujograma se relaciona con estructuras que definen el tipo de operación que representa el elemento. La estructura lógica de cada una de las operaciones del lengiajes es:

```
UNAFIGU = record
          NUMLINP,NUMLINS : byte    ;
          LINEASS,LINEASP : pointer ;
          ANCHO, ALTURA   : byte    ;
          RELLENO         : boolean ;
          XV,YV,XF,YF,PESO: byte    ;
          HAYPAR          : array [1..MAXPAR] of char    ;
          COORPAR         : array [1..MAXPAR] of integer;
          end;
```

La información que contendrá cada uno de los cmpos de una estructura de este tipo, será la siguiente:

NUMLINS: Indica cuantos puntos conforman el polígono que puede dibujarse con línea sólido, como parte del icono asociado a la operación.

NUMLINP: Indica cuantos puntos conforman el polígono que debe dibujarse con línea punteada, o el polígono que debe de rellenarse de color, como parte del icono asociado a la operación.

LINEASS: Es un apuntador que indica la dirección de memoria en donde se encuentra el arreglo de los pares ordenados correspondientes a cada uno de los puntos que conforman el polígono que debe dibujarse con línea sólida.

LINEASP: Es un apuntador que indica la dirección de memoria en donde se encuentra el arreglo de los pares ordenados correspondientes a cada uno de los puntos que conforman el polígono que debe dibujarse con línea punteada, o que debe rellenarse con collor.

ANCHO: Indica la anchura máxima del icono en unidades de dibujo, desde su centro hasta uno de sus extremos laterales.

ALTURA: Indica la altura máxima del icono, en unidades de dibujo, desde su centro hasta su extremo superior o inferior.

RELLENO: Este componente booleano indica si los campos lineasp y numlinp se refiere a un polígono relleno (si relleno es verdadero), o si se refieren a un polígono cuyo contorno debe ser dibujado con línea punteada (si relleno es falso).

XV y YV: Son las coordenadas x y y respectivamente en unidades de dibujo con respecto al centro del icono, desde donde debe salir una flecha sólida.

XF y YF: Son las coordenadas x y y respectivamente en unidades de dibujo con respecto al centro del icono, desde donde debe salir una flecha punteada.

PESO: Contiene un número que representa la prioridad de la operación. Conforme el número sea más grande, mayor será la prioridad que tenga la operación para ejecutarse en caso de que se encuentre en un nodo terminal.

HAYPAR: Es un arreglo de caracteres que definen el tipo de argumentos (parámetros) que debe contener la operación. Si el i-ésimo componente de este arreglo contiene un caracter "N", indicará que el i-ésimo argumento de la operación contendrá un nombre, que puede ser el nombre de la relación. Si por lo contrario contiene una "P", indicará que se trata de un parámetro.

COORPAR: Es un arreglo que contiene las coordenadas con respecto al centro del icono, en donde deben ser vizualizada cada uno de los MAXPAR argumentos de la operación.

Cada una de las anteriores estructuras se encuentran agrupadas en un arrglo llamado ARREFIG, que se define como:

```
ARREFIG: array  1..MAXFIG] of UNAFIGU;
```

donde MAXFIG es el número total de simbolos definidos definidos en LIDA

Los nodos además de tener asociado elementos del arreglo ARREFIG, poseen información relevante que define su posición visual en el flujograma relacionados con otros nodos del mismo. Podríamos decir, que las ligas o lineas que indican gráficamente el flujo de datos, así como. la manera y orden en cómo el flujograma será ejecutado, estan contenidas en estas estructuras, que se presenta a continuación:

```
ELEMENTO: record

          NUMFIGURA        : byte ;
          CADPAR           : array [1..MAXPAR ] of ^string;
          OPERANDO         : array [1..MAXOPDO] of APUCOMP;
          POSX,POSY        : integer;
          YACALC           : boolean;
          ESULTI           : byte;
          ARCHIT,ARCHIF    : ^string
          end;
```

donde

```
APUCOMP = record

          CX,CY   : byte;
          TIPOL   : (SOLIDA,PUNTEADA)
          end;
```

NUMFIGURA: Es el número de la posición (indice) que ocupa la operación asociada al nodo, dentro del arreglo ARREFIG.

CADPAR: Es un arreglo de apuntadores, cada uno señalaran las cadenas que se generan dinámicamente, y que conformarán los argumentos o parámetros de la operación asociada al nodo. Estas cadenas serán capturadas por el usuario.

OPERANDO: Es un arreglo que indica como un nodo en particular esta ligado a que otros nodos del flujograma. El arreglo está formado por componentes del tipo APUCOMP, cuya estructura posee el siguiente significado:

CX y CY: Son las coordenadas de x y y respectivamente, en "unidades matriciales" de un nodo al cual está ligado el nodo en cuestión.

TIPOL: Indica el tipo de liga que existe entre el nodo actual y el nodo con el cual se está relacionando. Este tipo de liga, además de tener una función operativa, también indica el tipo de línea con que se debe dibujar la flecha correspondiente entre ambos nodos. Los dos valores posibles que puede tomar el campo TIPOL son SOLIDA y PUNTEADA.

POSX y POSY: Son las coordenadas x y y en "unidades de dibujo" que corresponden a la posición donde debe colocarse el centro del icono en el momento de visualizarlo.

YACALC: Es un campo booleano que indica si el nodo ya ha sido calculado con anterioridad, dentro de una misma ejecución del flujograqma al que pertenece.

ESULTI: Dependiendo del valor de este campo, el sistema podrá determinar si se trata de un nodo terminal; es decir, un nodo del cual no sale ningún flujo de datos.

ARCHIT: Es un apuntador a una cadena de caracteres, que se genera dinámicamente para almacenar el nombre del archivo donde se arrojen los resultados de la operación asociada al nodo. Si la operación es un proceso de selección, entonces la cadena señalada por este apuntador indicará el nombre del archivo donde se almacenaron los registros que cumplieron con la condición de filtrado.

ARCHIF: Es un apuntador a una cadena de caracteres, el cual será generado dinámicamente para almacenar el nombre del archivo donde se arrojaron los registros que no cumplieron con la condición de filtrado en una operación de selección.

Los últimos cuatro campos de la estructura ELEMENTO están íntimamente relacionados con el método de ejecución del flujograma, por lo cual se explicará posteriormente su funcionamiento detallado.

Ahora procederemos a tratar con más detenimiento la forma en cómo se establecen las posiciones y los tipos de cada nodo del flujograma; así cómo, la definición de los vínculos que dan la conectividad y el flujo de datos entre ellos.

En principio, se tiene la matriz DIBUJO con cada una de sus entradas vacías

DIBUJO [i,j] = NIL , 1 <= i <= MAXR, 1 <= j <= MAXC

Esquemáticamente lo podemos representar según la figura siguiente 7.1.

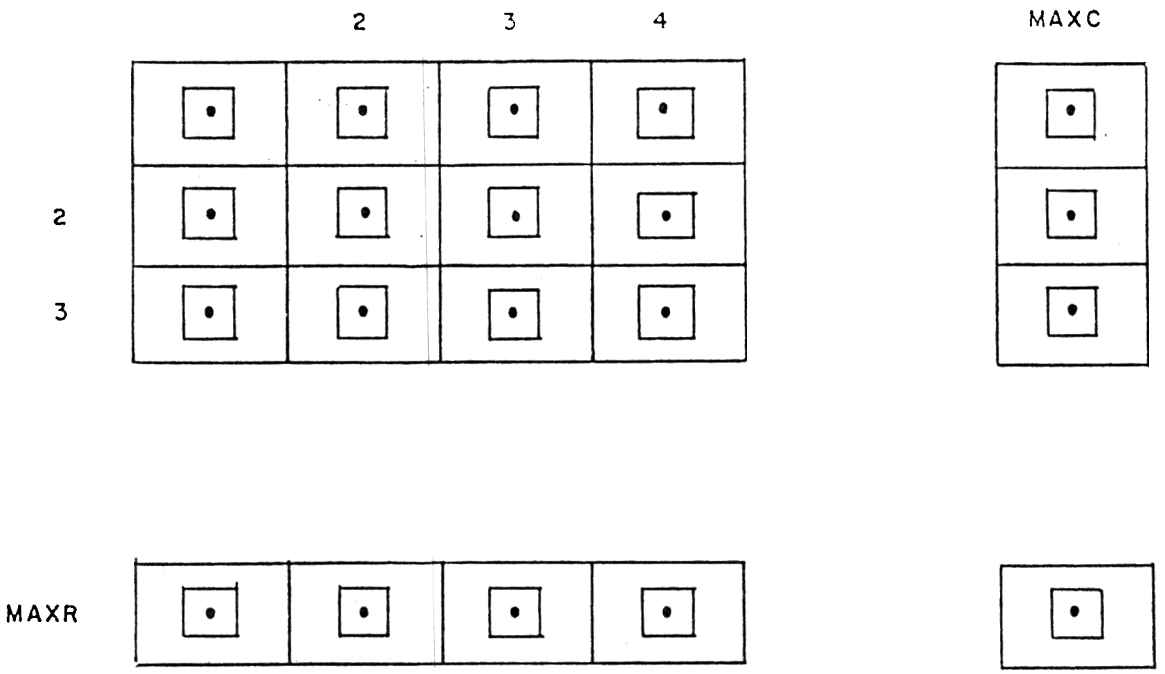


Fig. 7. MATRIZ DE APUNTADORES. ESTRUCTURA INTERNA DE UN FLUJOGRAMA

Ahora bien, en este momento es importante formalizar lo que hemos llamado "unidades de dibujo" y "unidades matriciales".

En el sistema se manejan internamente dos tipos de coordenadas: "de dibujo" y "matriciales". Las primeras se refieren a las que se utilizan para fines de visualización, considerando toda el área bidimensional para fines del dibujo del flujograma. Esta área de dibujo es una matríz de puntos, en terminos de los cuales se dan todas las dimensiones de los iconos, las posiciones que éstos ocupan dentro de tal área y los lugares donde deben visualizarse las flechas, así como, los parámetros de cada uno de los nodos.

Por otro lado, las coordenadas "matriciales" se refieren especificamente a las entradas del arreglo bidimensional DIBUJO, y son utilizadas para acceder a cada uno de sus elementos. De hecho, las ligas entre los nodos del flujogramase establecen mediante este tipo de coordenadas.

Cada que se mueve el cursor de edición del flujograma, se desplaza una cantidad preestablecida de puntos de dibujo, tanto vertical como horizontalmente, tomando una posición que se refiere dentro de la matríz de DIBUJO. La primera posición que puede tomar el cursor (esquina superior izquierda), corresponde al elemento (1,1) de la matriz; si posteriormente el cursor se mueve hacia la derecha, entonces la nueva posición se referirá al

elemento (1,2), independientemente del número de unidades de dibujo que se hayan desplazado visualmente entre una posición y otro. De esta manera cuando el usuario decide insertar un nuevo elemento al flujograma, son tomadas en cuenta las "coordenadas matriciales" actuales del cursor para determinar que apuntador es el que debe señalarlo.

En la figura 7.1 cada uno de los puntos representa un apuntador de la matriz. Cuando se inserta un nuevo elemento al flujograma, el apuntador correspondiente es dirigido hacia la localida de memoria donde se genera dinámicamente tal elemento. bajo la estructura ELEMENTO citada anteriormente. Como se ejemplifica en la figura 7.2.

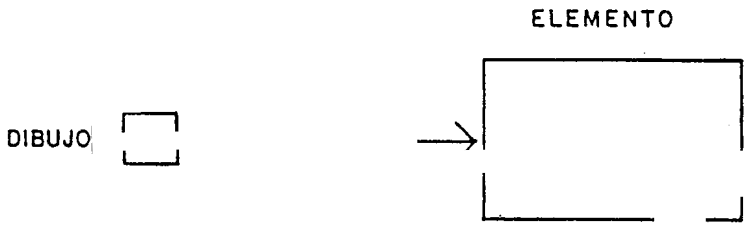


Fig. 7.2 GENERACION DE LA ESTRUCTURA ELEMENTO EN ALGUN NODO DE LA MATRIZ

Cuando el usuario inserta un icono en el flujograma, antes lo elige del menú seleccionando realmente una posición que ocupa la operación dentro del arreglo ARREFIG. Si n es el número de la operación seleccionada, entonces internamente se harán las asignaciones

```
new (DIBUJO [i,j] );
DIBUJO [i,j] ^.NUMFIGURA := n;
```

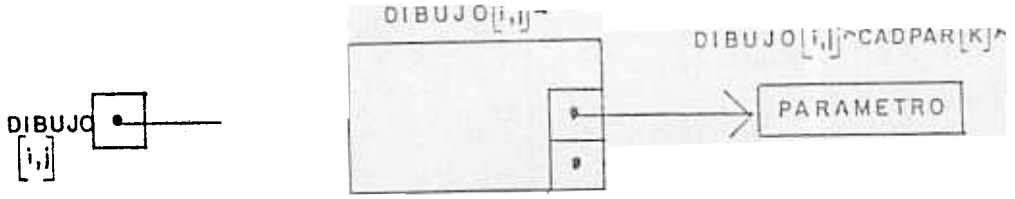
Con ello se podrá accesar toda la información necesaria para dibujar el icono correspondiente al elemento (i,j) del flujograma, así como aquella que indicará el comportamiento de la operación.

Una vez insertado el nodo al flujograma, se le asignan las "coordenadas del dibujo" (xd,yd) de la posición visual del cursor

```
DIBUJO [i,j  ^.POSX := xd;
DIBUJO [i,j  ^.POSY := yd;
```

con lo cual se sabrá en que lugar se deberá aparecer el centro del icono.

Posteriormente, el usuario podrá introducir los argumentos o parametros de la operación. Para ello se utiliza el arreglo de apuntadores llamado CADPAR. Si el k-esimo elemento del arreglo CADPAR es editado, entonces se genera un espacio en la memoria donde será almacenado. Gráficamente esta acción podria representarse según la figura 7.3.



Flg. 7.3 GENERACION DE ARGUMENTOS DE LA OPERACION ARREGLO DE APUNTADORES CADPAR

Por último el usuario deberá indicar la relación existente entre los elementos del flujograma a través de las flechas. Si existe una flecha que parte del elemnto (i,j) de la matriz al elemento (r,s), significará que hay un flujo de datos. Los componentes de la estructura ELEMENTO que permite definir internamente la relación anterior son cada uno de los MAXOPDO integrantes del arreglo OPERANDO. Cada uno de esos integrantes poseen a su vez tres campos: los campos CX y CY indicarán las "coordenadas matriciales" del elemento hacia donde se dirige la flecha, el campo TIPOL señalará el tipo de línea que se ha seleccionado para relacionar ambos elementos. A saber estan predefinidos dos tipos de linea: SOLIDA Y PUNTEADA.

Si tomamos el ejemplo donde se ha decidido trazar una linea sólida desde el elemnto (i,j) hasta el elemnto (r,s), internamente se harán las siguientes operaciones:

```
DIBUJO [r,s] ^.OPERANDO [k].TIPOl := SOLIDA;
DIBUJO [r,s] ^.OPERANDO [k].CX    := i      ;
DIBUJO [r,s] ^.OPERANDO [k].CY    := j      ;
```

donde m es le número del primer integrante vacio encontrado en el arreglo OPERANDO. Dado que este último arreglo tiene un máximo MAXOPDO integrantes, entonces el límite de flechas que pueden llegar a un nodo estarán dado por tal número.

Como se puede observar, la asociación de los elementos se realiza al revez de como el usuario lo indica y lo visualiza, debido al método de ejecución de los flujogramas.

La figura 7.4 muestra ambas versiones de una relación existente entre dos elementos.

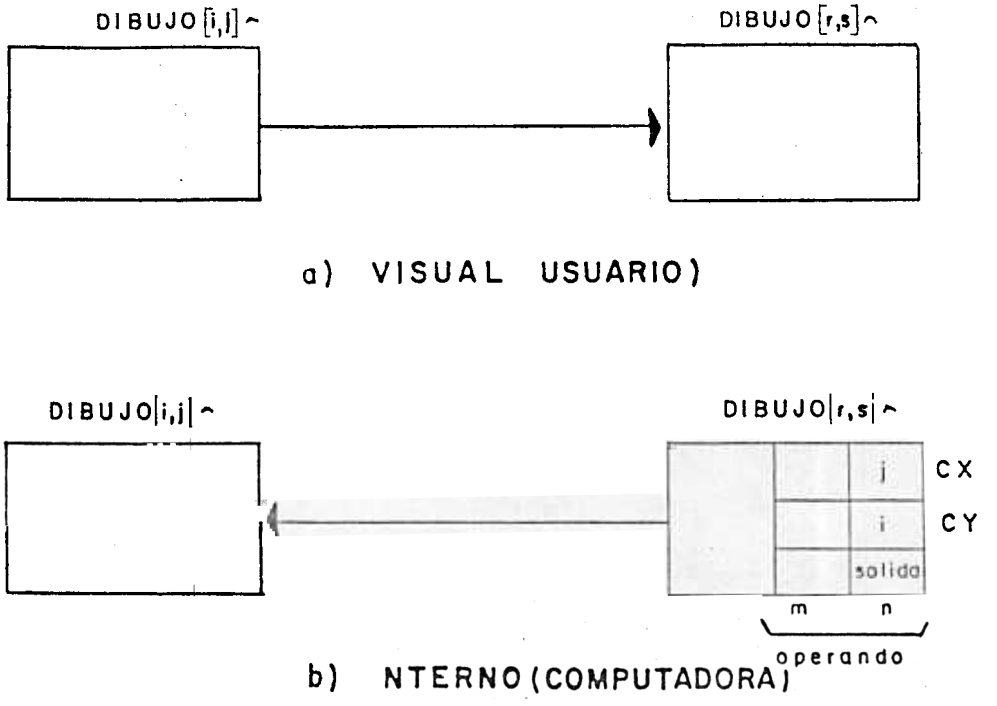


Fig 7.4 ASOCIACION DE ELEMENTOS: DOS REPRESENTACIONES

Si posteriormente se decidiera que el elemento (g,h) hay un flujo de datos hacia el (r,s), entonces se ocuparia otro de los integrantes del arreglo OPERANDO del elemnto (r,s).

2 METODO DE EJECUCION DE UN FLUJOGRAMA.

El método de ejecución de un flujograma podríamos decir que es un método "de abajo hacia arriba", basandose en la premisa de que para resolver la operación, necesito conocer el resultado de las operaciones inmediatas anteriores. De esta manera el algoritmo es sencillo y es aplicado de manera recursiva a cada uno de los nodos, salvo pequeñas variantes asociadas a la naturaleza de la operación involucrada.

Un ejemplo que muestra el funcionamiento del método, es dado por una operación aislada, cómo es la unión de dos relaciones que arrojan una tercera como resultado. Mostrado en la figura 7.5

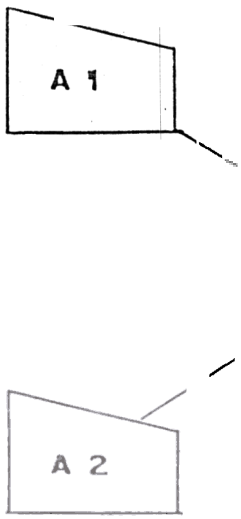


Fig. 7.5 EJEMPLO PARA MOSTRAR EL FUNCIONAMIENTO DEL METODO

En este caso la ejecución se iniciará en el nodo terminal, es decir, el que indica la definición de la relación A3. Como ésta es resultado de una operación, entonces se procede a ejecutar el nodo inmediato anterior, que corresponde a una unión; pero este no puede arrojar un resultado hasta conocer los datos provenientes de las operaciones anteriores, por lo cual nuevamente se procede a pasar "hacia arriba". Debido a que en este caso existen dos nodos anteriores, entonces se ejecutará el correspondiente al señalado por el primer integrante del arreglo OPERANDO; es decir, OPERANDO [1] siguiendo con OPERANDO [2], y así sucesivamente hasta completar los requerimientos de la operación asociada al nodo en cuestión. Supongamos que para nuestro ejemplo, primero se ejecuta el nodo que indica la definición de la relación A1, como éste ya no tiene un nodo antecesor, entonces se procede a extraer la información, previamente almacenada en ella, pasándola "hacia abajo" al nodo de la operación de unión, donde se procede a ejecutar la operación, que define de igual forma la relación A2. Como ahora ya se tienen todos los elementos para realizar la unión, se procede a ejecutarla, completando la operación del nodo y pasando la relación resultado hacia el nodo terminal, donde se procederá finalmente a copiar los resultados en la relación A3.

Por lo tanto, la manera de operar de un nodo se puede resumir en los tres pasos siguientes:

a Tomar los nombres de las relaciones arrojadas por las operaciones inmediatas anteriores,
b) Operar con tales relaciones, y
c) Pasar el nombre de la relación resultante hacia los nodos subsecuentes.

Es en esta etapa en donde los campos YACALC, ESULTI, ARCHIT y ARCHIF de la estructura ELEMENTO toman participación. Cuando una operación ha sido realizada en su totalidad, entonces en su nodo correspondiente se hará la asignación DIBUJO [I,J] ^.YACALC := TRUE, que indica que la operación ya ha sido calculada y ha producido un a relación resultante. Si posteriormente se hace referencia al nodo en cuestión se sabrá que no es necesario ejecutarlo. Por otra parte, las cadenas de caractéres apuntadas por los componentes ARCHIF y ARCHIT se genéran dinámicamente. Dichas cadenas contendrán los nombres de las relaciones que se generáron como resultado de la operación, de tal manera que puede hacerse referencia a ellas en caso en caso de ser necesario en una operación posterior. Recordemos que la única operación que puede arrojar dos relaciones resultantes es la selección, con salida verdadera (campo ARCHIT) y salida falsa (campo ARCHIF). En la figura 7.6 se ilustra lo anteriormente mencionado.

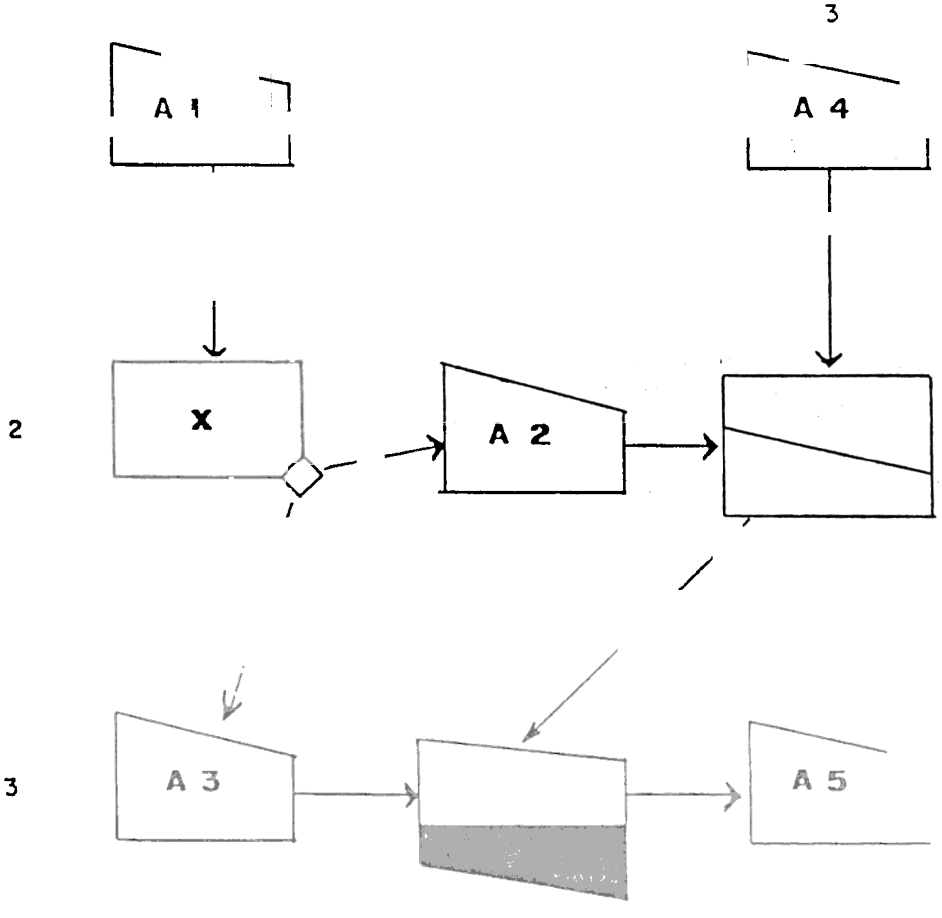


Fig. 7.6 EJEMPLO USADO PARA MOSTRAR LA GENERACION DE RESULTADOS INTERMEDIOS.

De acuerdo al método de ejecución se empezará por el nodo (3,3) pasando recursivamente "hacia arriba" por los nodos (3,2), (3,1), (2,1) y (1,1), el cual se resuelve al definir la relación A1. En este punto se hacen las asignaciones

```
DIBUJO [1,1] ^.YACALC := TRUE;
new ( DIBUJO [1,1] ^.ARCHIT )  Línea existente entre (2,1) y
                               (1,1) es de tipo sólido);
DIBUJO  1,1   .ARCHIT := 'A1';
```

y se procede a regresar "hacia abajo" el nombre de la relación resultante (en este caso A1). Así el nodo (2,1) ya tiene el nombre de la única relación que necesita para operar, arrojando como resultado dos relaciones, de las cuales sólo puede determinar el nombre de las que contendrá los registros que no cumplan con la condición y en el caso alternativo el sistema asignará el nombre.

```
DIBUJO [2,1] ^.YACALC := TRUE;
new (DIBUJO [2,1] ^.ARCHIT)   ;
new (DIBUJO [2,1] ^.ARCHIF)   ;
DIBUJO [2,1] ^.ARCHIT^ := 'ZN!';  Nombre asignado por el sistema
DIBUJO  [2,1] ^.ARCHIF^ := 'A3';  Nombre conocido por la  vista
                                  previa al nodo (3,1)).
```

Realizado lo anterior el nombre de la relación A3 es pasado al nodo (3,1), completando la operación definiendo una relación, con las siguientes asignaciones:

```
DIBUJO [3,1] ^.YACALC := TRUE ;
new (DIBUJO [3,1] ^.ARCHIT)   ;
DIBUJO [3,1] ^.ARCHIT^ := 'A3';
```

y pasando el nombre A3 al nodo (2,3)

Como este último requiere de dos relaciones para operar, se analiza "hacia arriba" el nodo (2,2) que es la definición de la relación A2, y que está sujeta al resultado del nodo (2,1). Al visitar este último, se encontrará que DIBUJO [2,1] ^.YACALC = TRUE, por lo que la operación no es necesario realizarla. Solo se procederá a regresar al nodo (2,2) el nombre de la relación de la salida verdadera de la selección, cuyo nombre se localiza en DIBUJO [2,1] ^.ARCHIT^ (ZN!). De vuelta en el nodo (2,2), se procederá a generar una relación con nombre A2, que contendrá la misma información que la ZN!, y se harán las asignaciones

```
DIBUJO [2,2] ^.YACALC := true ;
new (DIBUJO  2,2] ^.ARCHIT )  ;
DIBUJO [2,2] ^.ARCHIT^ := 'A2';
```

enviando su nombre hacia el nodo (2,3).

De éste se pasará nuevamente "hacia arriba" al nodo (1,3), de donde se obtendrá el nombre de la relación A4, de tal manera que la operación del nodo (1,3) podrá realizarse. Como puede

observarse el nodo posterior (visualmente) al (2,3) corresponde a una operación con relaciones; por lo que, al resultado de la operación en (2,3) se le asignará el nombre (ZN+1!).

```
DIBUJO [2,3]^.YACALC := true    ;
new (DIBUJO [2,3] ^.ARCHIT )    ;
DIBUJO [2,3]^.ARCHIT^ := Z(N+1)!;
```

pasando este nombre hacia el nodo (3,2), donde se realizará su operación dándole el nombre A5 a la relación resultante, nombre que conocía con la visita previa al nodo (3,3).

Finalmente, el nodo (3,3) recibirá el nombre A5, y como éste corresponde con el propio de su definición, no es necesario realizar la copia, finalizando la ejecución del flujograma. En la figura 7.7 tenemos un ejemplo de un flujograma que cuenta con más de un nodo terminal.

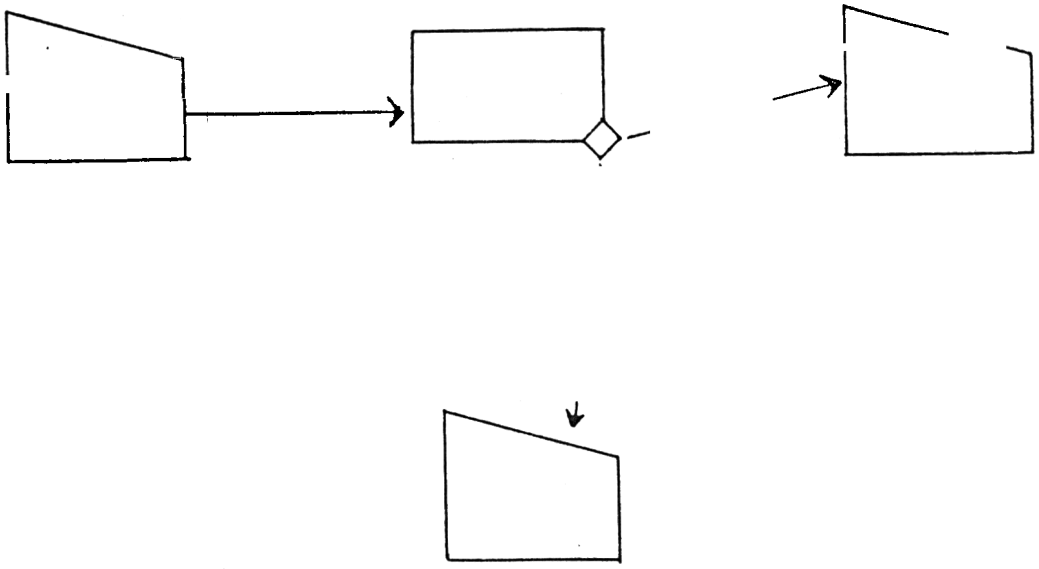


Fig. EJEMPLO DE FLUJOGRAMA QUE CUENTA CON MAS DE UN NODO TERMINAL

Para iniciar la ejeccución de un flujograma, debe seleccionarse el nodo terminal con mayor prioridad, ejecutando toda la rama del flujograma involucrada. Finalizando ésto, se procederá a seleccionar otro nodo terminal con menor o igual prioridad, aplicándosele el mismo proceso que al anterior. De esta manera se deberá cubrir totalmente los nodos terminales del flujograma.

Las prioridades de los nodos estan dadas por el tipo de operación asociada que se encuentra en el arreglo ARREFIG de la manera siguiente:

```
PRIORIDAD_I_J := ARREFIG   DIBUJO  I,J]^.NUMFIGURA   .PESO;
```

Al momento de recibirse la orden de ejecutar un flujograma, primero se revisan todos los nodos que componen el flujograma y

cuando se encuentra uno que es terminal se verifica que su operación asociada permita tal situación. De ser así, las coordenadas matriciales del nodo serán introducidas a un árbol binario ordenado de acuerdo al peso (prioridad) del nodo. Una vez revisados todos los nodos del flujograma, el árbol binario contendrá las coordenadas matriciales de todos los nodos terminales y sólo bastará recorrerlo en orden.

Hasta este momento hemos tratado el método de ejecución de un flujograma en su orden de recorrido, sólo falta aclarar la forma en que se ejecuta cada una de las operaciones en los nodos. La ejecución de un nodo difiere esencialmente en el tipo de cada operación, que se hace notar mediante:

* El número de operandos necesarios para realizar la operación.
* El número de relaciones que genera la operación.
* El número y tipo de argumentos o parámetros que requiere la operación.
* Un indicador que nos permite saber si la operación es terminal.

Cuando la operación de un nodo ya posse todos los elementos para trabajar, se procede a armar una cadena en lenguaje ISBL extendido, lenguaje intermedio de LIDA. El diagrama 7.8 muestra los pasos para ejecutar la operación de un nodo.

Retomando el ejemplo de la figura 7.6 tendremos que LIDA generará las cadenas de la manera siguiente:

NODO	CADENA
(2,1)	Z1!= A1:X, A3
(2,2)	A2 = Z1!
(2,3)	Z2!= A2 + A4
(3,2)	A5 = A3.Z2!

Es importante mencionar que las cadenas generadas a partir de dos flujogramas visualmente iguales pueden diferir. En el caso del ejemplo anterior, las cadenas corresponden:

Si se considera que el nodo (2,1 primero se ligó al (3,1) y luego al (2,2).

Que el nodo (2,2) se ligó primero al (2,3) y posteriormente el nodo (1,3) al (2,3).

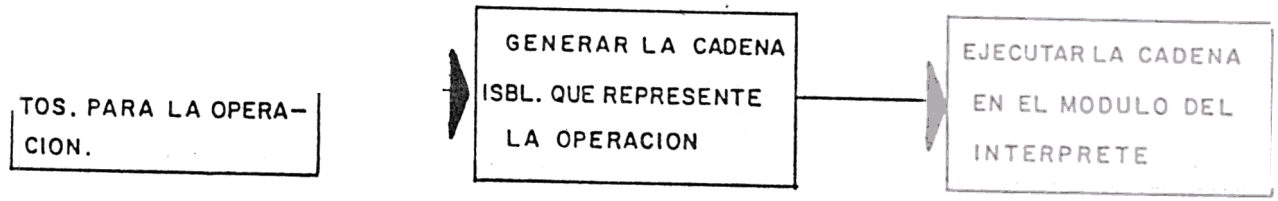
Y que el nodo 3,1 fué ligado primero al (3,2 y luego el (2,3) al (3,2).

Si se hubiese ligado primero el nodo (2,3) al (3,2) y posteriormente el (3,1) al (3,2), las cadenas generadas hubieran sido

NODO	CADENA
(2,1)	A2 = A1:X,z1
(2,3)	Z2!= A2+A4
(3,1)	A3 = Z1!
(3,2)	A5 = Z2!.A3

como puede observarse no altera el resultado final de la ejecución.

Para generar adecuadamente la cadena en ISBL extendido de la operación de un nodo, se hace uso de los argumentos de la operación, almacenados en las localidades de memoria apuntadas por los integrantes del arreglo CADPAR.



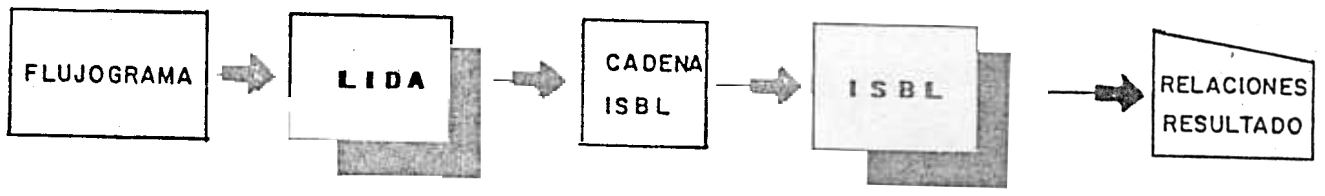


Fig. 7.8 DE UN FLUJOGRAMA

7.3 ALGORITMO DE EJECUCION

PROCESO PRINCIPAL;

Se considera un árbol binario, cuyos elementos son del tipo NODOTREE:

```
NODOTREE = ^NODARBOL;

NODARBOL = record
           POSX,      {Columna en la matriz del nodo terminal}
           POSY,      {Renglón en la matriz del nodo terminal}
           PRIO       {Prioridad de ejecución del nodo     }
                :byte;
           IZQ,       {Apuntador a la rama izquierda         }
           DER        {Apuntador a la rama derecha           }
                :NODOTREE
           end;
```

La variable árbol es del tipo NODOTREE y representa el apuntador al nodo padre del árbol binario.

En la rama izquierda se almacenan las coordenadas de los nodos que tengan prioridad de ejecución menor o igual y en la derecha los que tengan prioridad superior.

PASO 1: Para toda i,j, 1 <= i <= MAXR y
1 <= j <= MAXC,
se relizan los pasos 1.1 y 1.2;

PASO 1.1: Si el nodo (i,j) es terminal, entonces se verifica que la operación correspondiente permita tal situación. De ser asise realiza el paso 3, de lo contrario, se envia un mensaje de error y finaliza la ejecución.

1.2: Se procede a insertar las coordenadas matriciales (i,j) del nodo y su prioridad, señalada por
PRIORIDAD_I_J := ARREFIG [DIBUJO[i,j]
^.NUMFIGURA].PESO
en un nuevo elemento de ARBOL:
INSERTA_NODO_EJ (ARBOL, i, j, PRIORIDAD_I_J);

PASO 2: Se recorre el árbol binario, partiendo del apuntador ARBOL, con la llamada: RECORRE_ARBOL_EJ (ARBOL):

PROCESO INSERTA_NODO_EJ (PTR, REN, COL, PRIORIDAD);

PTR es un apuntador de tipo NODOTREE;
COL es una columna de la matriz DIBUJO donde se localiza el nodo a insertar en el árbol;

es el renglón de la matriz DIBUJO donde se localiza el nodo a insertar en el árbol;

PRIORIDAD indica la prioridad de ejecución de la operación asociada al nodo (REN,COL) de la matriz DIBUJO;

PASO 1: Si PTR no apunta a ningún elemento del árbol, entonces se realiza el paso 1.1, de lo contrario se realiza el paso 1.2;

PASO 1.1: Se genera un nuevo elemento en el árbol, que será señalado por PTR, y se asignan los valores:
PTR^.POSX := COL; PTR^.POSY := REN;
PTR^.PRIO := PRIORIDAD;
dando por terminado el proceso INSERTA_NODO_EJ;

PASO 1.2: Si PRIORIDAD <= PTR^.PRIO entonces el nuevo nodo debe generarse en la rama izquierda del elemento señalado por PTR: INSERTA_NODO_EJ (PTR^.IZQ,REN,COL,PRIORIDAD); de lo contrario debe generarse en su rama derecha:
INSERTA_NODO_EJ (PTR^.DER, REN, COL, PRIORIDAD);

FIN.

PROCESO RECORRE_ARBOL_EJ (PTR ;

PTR es un apuntador de tipo NODOTREE;

PASO 1: Si PTR apunta algún elemento del árbol; entonces se procede a realizar los pasos 1.1 al 1.3 de lo contrario, el proceso RECORRE_ARBOL_EJ se dá por terminado.

PASO 1.1: Se recorre la rama del árbol correspondiente a los nodos con prioridad mayor de ejecución al indicado por PTR: RECORRE_ARBOL_EJ (PTR^.DER);

PASO 1.2: Se procede a realizar el recorrido del flujograma, partienedo del nodo indicado por las coordenadas en el elemento del árbol apuntado por PTR:
TRADUCE_DIBUJO (PTR^.POSY, PTR^.POSX);

PASO 1.3: Se recorre la rama del árbol correspondiente a los nodos con prioridad igual o inferior de ejecución al del apuntado por PTR:
RECORRE_ARBOL_EJ (PTR^.DER);

FUNCION TRADUCE_DIBUJO (CY, CX);

CX es la columna de la matriz DIBUJO, correspondiente al nodo que va a ser ejecutado;

CY es el renglón de la matriz DIBUJO, correspondiente al nodo que va a ser ejecutado.

La función TRADUCE_DIBUJO regresa el nombre de la relación donde se almacene el resultado de la operación indicada por el nodo DIBUJO [CY, CX];

PASO 1: Si el nodo (cx, cy) ya ha sido calculado en un recorrido anterior, entonces se procede a regresar el nombre de la relación donde se almacenó el resultado:
TRADUCE_DIBUJO := DIBUJO [cy, cx]^.ARCHIT^; o
TRADUCE_DIBUJO := DIBUJO [cy, cx]^.ARCHIF^; según sea el caso y se finaliza la función. De lo contrario se realiza el paso 2;

PASO 2: De acuerdo a la operación asociada al nodo (cy, cx) de la matriz, se determina el número de operandos NUMOPDS que se requieren para producir un resultado;

PASO 3: Se determinan los nombres de las NUMOPDS relaciones, con las cuales se debe operar para producir un resultado:
Para toda i, 1 <= i <= NUMOPDS, debe calcularse el nombre REL [i], donde
REL [i]:=TRADUCE_DIBUJO (DIBUJO[cy,cx]^.OPERANDO[i].cy
DIBUJO[cy,cx]^.OPERANDO[i].cx);

PASO 4: Se determina el número NUMPAR de argumentos que requiere la operación de acuerdo a su naturaleza.

PASO 5: Para toda i, 1 <= i <= NUMPAR, se define el argumento PAR[i], donde: PAR[i] := DIBUJO [cy,cx]^.CADPAR[i].

PASO 6: Con los nombres de las relaciones REL y los argumentos PAR, se forma una cadena de caracteres CAD_EJEC, según corresponda a la sintaxis de la operación dada en el lenguaje ISBL extendido.

PASO 7: La cadena CAD_EJEC es ejecutada en el módulo del lenguaje anfitrión. De acuerdo a nuestro sistema, esto se hace siguiendo los mismos procedimientos aplicados a una cadena introducida por el usuario en el interprete ISBL.

PASO 8: El nombre de la relación resultante de la ejecución de la cadena CAD_EJEC, es asignado a la función TRADUCE_DIBUJO.

FIN.

7.4 DIAGRAMA PARA LA INTERPRETACION DE CADENAS DE ISBL.

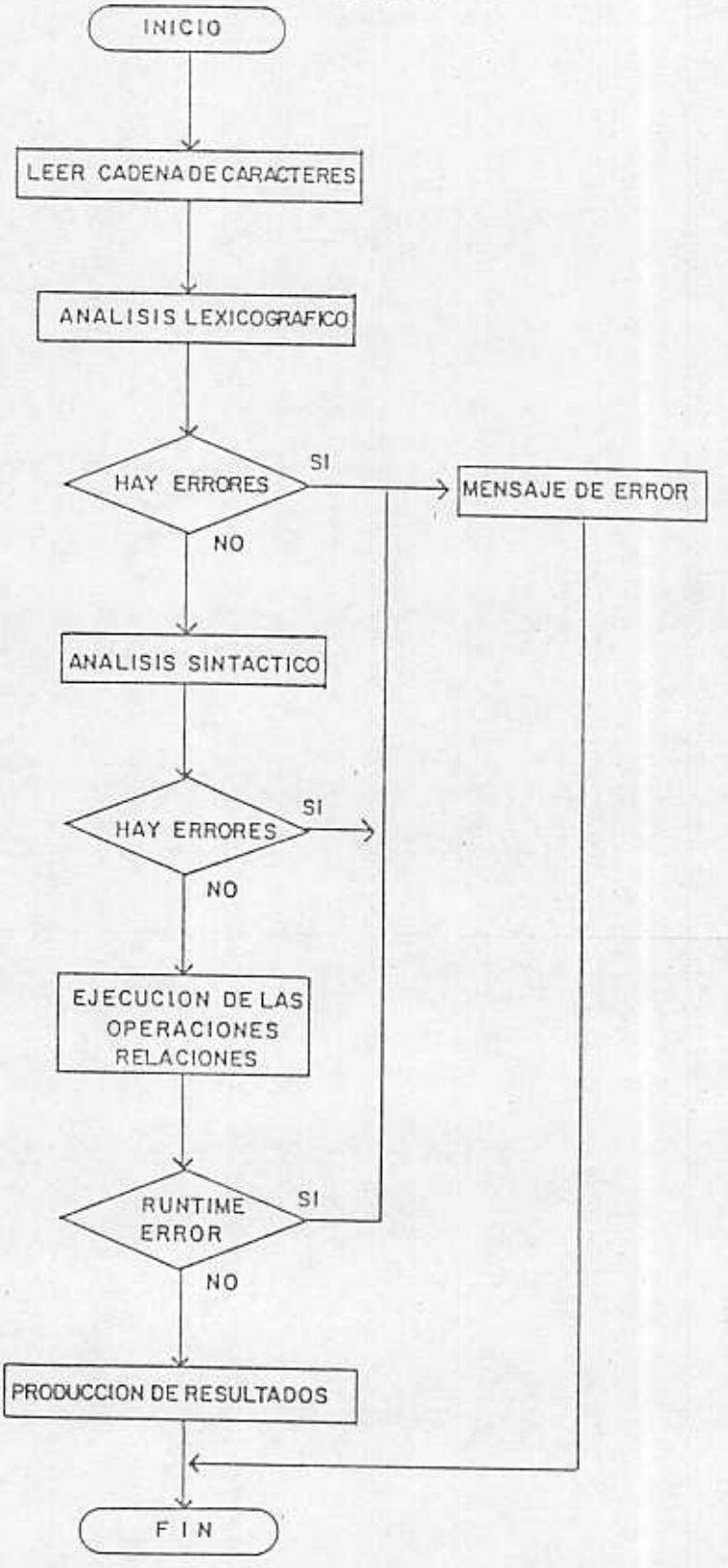


Fig. 7.9 DIAGRAMA DE FLUJO PARA LA INTERPRETACION DE CADENAS ISBL

CONCLUSIONES

El objetivo de este capitulo es dar los últimos comentarios acerca trabajo y conclusiones generales.

La contribución principal de LIDA es la de un lenguaje icónico que puede plantear problemas con diagramas denominados flujogramas y, a partir de estos la máquina efectúa transformaciones automáticas para encontrar soluciones.

LIDA es muy visual debido a que maneja fundamentalmente iconos en diagramas denominados flujogramas que se editan a manera de CAD para la programación. Los iconos son entidades geométricas que forma parte del alfabeto para formar flujogramas, que además utilizan texto como parte integral del lenguaje.

El usuario y/o programador no requiere necesariamente de conocer la estructura y tratamiento interno de los objetos. LIDA en la ejecución de flujogramas crea nuevos objetos sin dar cuenta al programador de la estructura y tipos de los datos. Sin embargo, los usuarios tienen la facilidad de definir sus esquemas y tener una documentación con el lenguaje de definición de datos; que igualmente gráfico, tiene el enfoque alterno de esqueleto de tabla con manejo de objetos de datos como principal representación.

LIDA es un excelente lenguaje de consulta de tipo gráfico, por medio del cual es posible formular consultas con los flujogramas a una base de datos. Acerca de la denominación como lenguaje visual de consulta aseguro que es uno de los que tienen más contenido gráfico; ver tabla comaprativa de sistemas icónicos sección 3.8. SDMS es un sistema eminentemente gráfico, cuyos datos tienen su representación directa a iconos, a diferencia de otros basados en formas o esqueletos de tablas. Solo algunos lenguajes (LID, GUIDE, etc.) formulan sus consultas en base de diagramas, considerando el modelo tradicional de entidad-relación. LIDA es una alternativa nueva como lenguaje visual de consulta que usa iconos y una representación de diagrama diferente. Con esta forma es posible tener despligue parciales de vistas a una consulta.

Con respecto al modelo de flujo de datos, LIDA omite descripciones de control de flujo para dar a los flujogramas una programación imperativa y representación en paralelo. Abstracción procedural se determina mediante la definición de iconos que encapsulan procesos y operadores. Los primeros instancían repetitivamente formas algebráicas sobre un conjunto de objetos de datos, evitando explicitamente la declaración de ciclos. Los segundos son operadores unarios o binarios que dan como resultado un objeto de relación. Las bifurcaciones las lleva a cabo LIDA con su constructor condicional.

De esta forma resumimos:

a) LIDA es un lenguaje eminentemente visual, debido a que esta constituido de iconos y diagramas denominados flujogramas.

b) LIDA se sustenta en el modelo de datos relacional

c) LIDA es un lenguaje de flujo de datos.

d) El sistema de LIDA tiene un módulo de descriptor que sirve para definir datos y esquemas de relación.

e) El sistema LIDA tiene un captador para construir su base de datos.

f) LIDA genera código intermedio de cadenas basadas en el lenguaje ISBL extendido.

El desarrollo de este prototipo ha permitido una experiencia para mejorar el sistema y proponer algunos proyectos nuevos.

Modelos Semánticos.

Un tema de interés que estamos iniciando es el de desarrollar modelos que capturen más semántica de los datos, iniciando con la aplicacio'n del modelo de Entidad-Relación. Particularmente, en los sistemas CASE el modelo de Chen se ha usado como un estandar para el diseño conceptual de base de datos. Su naturaleza gráfica en terminos de diagramas resulta adecuada como subsistema para el modelado de datos en LIDA (tesis de maestría por Raúl Hernández Stefanoni). Como parte del trabajo, ya se tiene desarrollado el módulo de normalización basado en el método de síntesis del algoritmo de Berstein [GONZ90].

Extensiones a otros modelos como son redes semánticas y marcos resulta atractivo, con la idea también, de incorporar operadores y álgebras analogas a la relacional. Sin embargo, un problema significativo es la eficiencia en la funcionalidad de los sistemas que incluyen en este contexto base de datos y de conocimiento.

Funcionalidad y Técnicas de Implementación.

Al respecto estamos estudiando una nueva propuesta de modelo físico que incluye una generalización del Descriptor de Archivos en: Diccionario de Datos, Descriptor de Esquemas y Estructuras de de Organización y Acceso; todas para determinar la semántica y la representación de la base de datos.

El primero define la semántica a más bajo nivel con los tipos, longitud y llaves. Este sistema interactúa con el Lenguaje de Definición de Datos. El segundo se construye de

acuerdo al diseño conceptual, en caso particular a partir del diagrama de entidad relación. Y el tercero proporciona diferentes estructuras de organización y acceso como son: B* Tree, empatamiento parcial, descriptor de código independiente, archivos invertidos; asociando los mecanismos de acceso.

Interfase Hombre-Máquina

Una contribución fundamental de LIDA es su ambiente gráfico de trabajo; sin embargo, para tener un sistema completo con este enfoque es necesario tener todas las facilidades al respecto. La primera consiste en el uso del ratón como un medio para la edición de los flujogramas, dando facilidades al sistema con el manejo de iconos, ventanas y textos. La segunda corresponde a una salida por medio de gráficador de pluma, con la finalidad d tener una documentación de los flujogramas.

El color aún no ha sido explotado suficientemente en relación a la participación del significado de los iconos y diagramas. Forma y color son dos elementos que pueden ser combinados para elaborar representaciones más complejas.

Los flujogramas son excelentes para representar una simulación de los proceso que se llevan a cabo sobre la información, teniendo la oportunidad de visualizar como fluye la información con vistas parciales de resultados. Se tiene pensado implementar un ambiente animado en LIDA con iconos que son valvulas y/o semáforos para suspender procesos o presentar vistas de resultados parciales.

El Sistema LIDA y CASE.

Debido a las caracteris ticas gráficas de LIDA y su representación funcional dentro del ambiente de flujo de datos, puede ser usado para el diseño de sistemas generando códigos de lenguajes. Actualmente, sólo se tiene la generación de código de cadena intermedio; sin embargo, se pretende generar códigos como SQL para sistemas de base de datos y Pascal, Cobol y C como lenguajes de programción.

Además relccionado al tema anterior es necesario tener buenos documentadores gráficos y para código.

Nuevos Lenguajes Visuales.

A partir de LIDA hemos estado diseñando otro lenguaje de tipo visual. LIPEC es el acróstico de Lenguaje Iconográfico para Programación Basado en Estructuras de Control, y es un lenguaje visual que para el desarrollo de programas considera la técnica de "top-down" acompañada de un refinamiento de paso a paso, en un proceso recursivo de construcción. En él intervienen dos fases principales: a) diseño del esqueleto de control y b) definición detallada de cada parte del esqueleto.

Partiendo de lo anterior, el objetivo es el diseñar programas con un modelo estructural de control de flujo y especificaciones de pseudocódigo. Tomando la idea de Glinert [GLIN85], acerca del paradigma de los bloques BLOX, el sistema consiste de iconos rectangulares que se ensamblan a manera de piezas de rompecabezas para construir una estructura de control, detallando en algunos de los modulos de forma recursiva una nueva gráfica de esqueleto de control (tesis de maestria de Noe Sierra en desarrollo)

Consultas Difusas y Operadores de Clasificación.

El objetivo en esta dirección es la implantación de operadores de clasificación y los que actuan sobre conjuntos difusos con la finalidad de tener un sistema más inteligente. Actualmente tenemos un sistema automático de transformación sobre conjuntos difusos [CHAP90], con la idea de tener un procesador de consultas difusas a base de datos (tesis de mastria Edgar Gonzalez A.). El interés de implantarlo al sistema LIDA dentro de un ambiente gráfico, nos motivo ha crear nuevos iconos con mayor semántica que sirvieran para declarar las consultas difusas. Usamos caras de Chernoff como iconos que representan operadores de clasificación [CHAP89], considerando que los razgos se mapean en variables lingüisticas y que se evaluan a través de la composición de las operaciones básicas sobre los conjuntos difusos: intersección, complemento, dilatación y concentración (tesis de licenciatura Catalina Zecua F. [ZECU91]).

Sistema Distribuidos y Procesamiento en Paralelo.

Una de las principales características de los sistemas basados en flujo de datos es su alta representación de paralelismo; sin embargo, para llevar a cabo un procesamiento de tal naturaleza, es necesario una máquina para tal efecto, o bien, un sistema distribuido que delegue la ejecución de tareas simultaneas. LIDA basado en el modelo de flujo de datos tiene posibilidades de procesamiento en paralelo, con un tratamiento inmediato de distribución de diversos procesos.

En principio, uno de los principales problemas en el modelo relacional ha sido encontrar una eficiente implementación, cuya dificultad principal ha consistido en que los operadores binarios: junta, unión, intersección y diferencia; requieren de preprocesados que ordenen relaciones para su uso. La fuerte modularidad de LIDA permite que consideremos operadores que se ejecuten indemendientemente, más aún, procesos mas complejos.

REFERENCIAS Y BIBLIOGRAFIA.

AIR 74 Air Force Materials Laboratory, AFSC, WPAFB, "Air Force Computer-Aided Manufacturing (AFCAM) Master Plan", (vol. II, App. A, y Vol. III) (Report *AFML-TR-74-104 disponible de DDG como AD 922-0411 y 922-171L), Julio 1974.

BALZ76 BALZER, R., N. GOLDMAN, y D. WILE, "Meta-Evaluation as a Tool for Program Understanding," Technical Report ISI/RR 78-69, Information Sciences Institute, University of Southern California, Enero 1978.

BALZ77 BALZER, R., "Correct and Efficient Software Implementation via Semi-Automatic Transformations. " USC/ISI Internal Rep., Information Science Institute, Univ. of Southern California, Marina del Rey. 1977.

BALZ78 BALZER, R., N. GOLDMAN, y D. WILE, "Informality in Program Specifications." IEEE Transactions on Software Engineering 4(2), Marzo 1978.

BARS76 BARSTOW, D., y E. KANT, "Observations on the Interaction between Coding and Efficiency Knowledge in the PSI Program Synthesis System," Proceedings of the Second International Conference on Software Engineering, Octubre 1976.

BARS77a BARSTOW, D., "A Knowledge-Based System for Automatic Program Construction." Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Agosto 1977.

BARS77b BARSTOW D, R., "Automatic Construction of Algorithms and Data Structures using a Knowledge Base of Programming Rules." Ph.D. dissertation, Stanford Univ., Stanford, Calif., 1977.

BARS77c BARSTOW, D. R., "A Knowledge Base of Organization for Rules about Programming". SIGART Newsl. (ACM) 63, (June 1977), 18-22.

BARS79a BARSTOW, D. R., "An Experiment in Knowlegde-Based Automatic Programming." Artif. Intell. 12-73-119. 1979.

BARS79b BARSTOW, D. R., "The Roles of Knowledge and Deduction in Program Synthesis." In Proceedings of the 6th International Joint Conference on Artificial Intelligence (Tokyo, Aug. 20-23). International Joint Council on Artificial Intelligence, Inc., Stanford, Calif., 1979. pp. 37-43.

BARS79c BARSTOW, D. R., "On Convergence Toward a Data Base of Programming Rules". Paper distributed at 2nd Program Transformation Workshop (Cambridge, Mass., Sept.). 1979.

BENI78 BENIGER, JAMES R. y ROBYN, DOROTHY L., "Quantitative Graphics in Statistics: A Brief History.", American Statistician, 32:1, 1978.

BLAZ73 BLAZER, R., "A Global View of Automatic Programing", In Proceedings of the 3rd International Conference on Artfi_ cial Intelligence (Stanford, Calif.). 1973. pp. 494-499.

BOBR74 BOBROW, D.G., y RAPHAEL, "New Programming Language for Artificial Inteligence Research.", Computing Surveys, 6(3), 1974.,pp. 153-174.

BORN81 BORNING ALAN, "The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory", ACM Transacción on Programming Languages and Systems 3, No. 4, 1981, 353-387.

BRAC77 BRACKETT, J. W. y G. L. McGOWAN., Applying SADT to Large System Problems", Software Phenomenology, U.S. Army Computer Systems Command. 1977. pp. 539-552.

BROD84 BRODIE, M. L., MYLOPOULOS, J. y SCHMIDT J. W. (eds.)., "On Conceptual Modeling: Perspective from Artificial Intelligence, Databases, and Programming Language.", Springer Verlag, New York, 1984.

CARD85 CARDELLI, L., y WEGNER, P., "On Understanding Types, Data Abstracction, and Polymorphismo.", ACM Computing Survey, 17, 4 (Dic. 1985), 471-522.

CHAP85 CHAPA V. SERGIO, "Herramienta para Consulta y Captura basadas en el Descriptor de Archivos.", Reporte Técnico No. 15, Serie Amarilla Investigación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, febrero 1985.

CHAP85a CHAPA V. SERGIO, "Herramienta de Consulta y Captura en base a Descriptores de Archivos.", II Congreso Internacional de Ingeniería en Comunicaciones Eléctricas y Electrónica, AMICEE, México, D.F., noviembre 1985. pp. 47-56.

CHAP86 CHAPA, VERGARA SERGIO V., "Definición del Modelo de un Lenguaje de Programación (Lenguaje de Flujograma).", Reporte Técnico No. 51, Serie Amarilla Investigación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, Noviembre 1986.

CHAP87 CHAPA, VERGARA SERGIO V., "Lenguaje de Flujograma para la Automatización de las Oficinas.", Memorias de las Conferencias de MEXICON 87 IEEE, Octubre de 1987, pp. 47-56.

CHAP89 CHAPA, VERGARA SERGIO V., "Lenguaje de Flujograma para la Consulta en Base de Datos (Un Estudio Comparativo).", Memorias de la Quinta Conferencia Internacional Las Computadoras en las Instituciones de Educación y de Investigación. UNISYS-UNAM, Noviembre 1989. (Por publicar).

CHAP89a CHAPA, VERGARA S. V., y ZECUA FERNANDEZ CATALINA., " Caritas como Representación Iconica para el Procesamiento de Consultas Difusas en el Lenguaje de Flujograma.", Memorias de la Quinta Conferencia Internacional Las Computadoras en las Instituciones de Educación y de Investigación. UNISYS-UNAM, Noviembre 1989. (Por publicar).

CHEN76 CHEN, PETER, "The Entity-Relationship Model- Toward a Unified View of Data", ACM Transactions on Database Systems, Vol 1, No. 1, Marzo 1976, pp. 9-36.

CINC78 CINCOM SYSTEMS, INC. 1978. Total Information System, The Next Generation of Software

CODA71 CODASYL DATA BASE TASK GROUP, 1971. Final report. In Conference on Data System Languages (Apr.). ACM, New York.

CODD70 CODD, E. F., "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, 13, 6, (Junio 1970), 377-387.

CURRY78 CURRY, G. A., "Programming by Abstract Demostration.", Disertacion PhD, Dept. de Ciencias de la Computación, Universidad de Washington, Tech. Report 78-03-02, Seattle Wash. 1978.

DAHL72 DAHL, O. J. y HOARE , C.A.R., "Hierarchical Program Structures", en Structured Programming, O. J. Dhal, E. W. Dijkstra y C. A. R. Hoare. New York: Academic Press, 1972, pp. 175-220.

DENE74 DENERT, E., FRANCK R., y STRENG W., "PLAN2D-Towards a Two-Dimensional Programming Language." Proc Fourth Gesellschaft fü Informatik Berlin, 1974 pp. 202-213. (Publicado por Springer Verlag, Berlin, as Vol. 26 de Lecture Notes in Computer Science).

DREY72 DREYFUSS HENRY, "Symbol Sourcebook", Mc Graw-Hill Book Company, New York, 1972.

ELLI80 ELLIS CLARENCE A. y NUTT GARY J., " Office Information Systems and Computer Science.", Computer Surveys ACM, Vol. 12, No. 1, marzo 1980, pp. 27-60.

ELLI69 ELLIS, T. O., HAEFNER J.F., y SIBLEY W. L., "The GRAIL Proyect: An Experiment in Man-Machine Communications", Rand tech. report RM-5999-ARPA, The Rand Corporation, Santa Monica, Calif. 1969.

FIEN77 FIENBERG, S. E., "Graphical Methods in Statistics." Technical Report No. 304, Dept. of Applied Statistics, University of Minnesota, 1977, pp. 44.

FOGG84 FOGG, DENNIS, "Lessons from a Living in a Database", Graphical Query interface.", ACM SIGMOD. 1984, pp. 100-106.

GANE79 GANE CHRIS y TRISH SARSON, "Structured Systems Analysis: Tools and Techniques", Prentice-Hall., Englewood Cliffs, New Jersey, 1979.

GLIN84 GLINERT, E. P. y TANIMOTO, S. L., "PICT: An Interactive, Graphical Programming Environment." Computer, Vol. 17, No. 11, Nov. 1984, pp. 7-25.

GREE75 GREEN, C., y BARSTOW, D. "Some Rules for the Automatic Synthesis of Programs.", In Advance Papers of the 4th International Joint Conference on Artificial Intelligence (Tbilisi, Georgia, USSR, Sept. 3-8). International Joint Council on Artificial Intelligence, Inc., Stanford, Calif., 1975.

GREE76 GREEN, C., "The Design of the PSI Program Synthesis System," Proceedings of the Second International Conference on Software Engineering, Octubre 1976.

GREE77 GREEN, C., "A Summary of the PSI Program Synthesis System". In Proceedings of 5th International Joint Conference on Artificial Intelligence (Cambridge, Mass.). M. I. T., Cambridge, Mass., pp. 380-381, 1977.

GREE78a GREEN, C., "The PSI Program Synthesis System 1978: an Abstract". In Proceedings of 1978 National Computer Conference (Anaheim, Calif., June 5-8), AFIPS Press, Reston, Va., pp. 673-674, 1978.

GREE78b GREEN C., y BARSTOW, D. "On Program Synthesis Knowledge". Artif. Intell. 10, 241-279.

GREE79 GREEN, C., GABRIEL, R. P., KANT, E., KEDZIERSKI, B. J., McCUNE, B. R., PHILLIPS, J. V., TAPPEL, S. T., y WESTFOLD, S. J., "Results in Knowledge Based Program Synthesis". In Proceedings of 6th International Joint Conference on Artificial Intelligence (Tokyo, Aug. 20-23). International Joint Council on Artificial Intelligence, Inc., Stanford, Calif., pp. 342-344, 1979.

GUIB78 GUIBAS, L., y D. WYATT, "Compilation and Delayed Evaluation in APL," Procedings of the Fifth ACM Symposium on Principales of Programming Languages, Enero 1978.

GUZM85 GUZMAN A. A., "The File Descriptor: Use of a Descriptive Tool to Retrive General Queries to Files.", Reporte Técnico No. 16, Serie Amarilla Investigación, Departamento de Ingenieria Eléctrica, CINVESTAV-IPN, mayo 10, 1985

HAMM77 HAMMER, M., ET AL., "A Very High Level Programming Language for Data Processing Aplications," Communications of the ACM, 20(II), Noviembre 1977.

HERO80 HEROT, F. CHRISTOPHER, "Spatial Management of Data", ACM Transactions on Database Systems, Vol. 5, Num. 4, Diciembre 1980, pp. 493-513.

HEWL78 HEWLETH-PACKARD Company HP300 Computer System General Information Manual, Santa Clara, Calif., Sept. 1978.

HOUS76 HOUSEL, B. C. y SHU, N. C., "A High-Level Data Manipulation Languaje for Hierarchical Data Structures.", Proc. Conf. Data Abstraction, Definition and Structure. Marzo 1976, pp. 155-168.

IBM75 IBM Information Management System/Virtual Storage (IMS/VS). General Information Manual, G 20-1260-3, IBM Corp.

JARK85 JARKE, M. y VASSILIOU, Y., "A Framework for Choosing a Database Query Language." ACM Computing Surveys 17, No 3, Septiembre 1985, pp. 313-340.

KANT77 KANT, E., "The Selection of Efficient Implementations for a High-level Language". In Proceedings of 4th ACM Symposium on Artificial INtelligence and Programming Languages (Rochester, N. Y., Aug. 15-17). SIGPLAN Not. (ACM) 12,8 (Aug.)/SIGART Newsl. (ACM) 64 (Aug.), 140-156, 1977.

KANT79 KANT, E., "A Knowledge-based Approach to Using Efficiency Estimation in Program Synthesis." In Proceedings of 4th ACM Symposium on Artificial Intelligence (Tokyo, Aug. 20-23). Intsernational Joint Council on ARtificial Intelligence, INc., Stanford, Calif., pp. 457-462, 1979.

KANT81 KANT, E., y BARSTOW, D. R., "The Refinement Paradigm: The Interaction of Coding and Efficency Knowledge in Program Synthesis". IEEE Trans. Softw. Eng. 7, 458-471. 1981

KITW84 KITAGAWA H., "Form Document Management System SPECDOQ-Its Arquitecture and Implementation.", Proc. Second ACM Conf. Office Information Systems, Junio 1984, pp. 132-142.

KNUT74 KNUTH, D. E., "Structured Programming with goto Statements". ACM Comput. Surv. 6, 4 (Dec.), 1974, 261-301.

KLEI70 KLEIN ROBERT, "Form and Meaning", Princeton University Press, New Jersey, 1970.

KRUS73 KRUSKAL, WILLIAM, "Visions of Maps and Graphs." Proccedings of the International Symposium on Computer assisted Cartography, 1973, pp. 27.

LACR80 LACROIX, M. y PIROTTE A., "User Interfaces for Database applicatons programming". Tech. Rep. Phillips MBLE Research Laboratory, Bruselas, Belgica, 1980.

LISK74 LISKOV, B. H. y ZILLES, S. N., "Programming with Abstract Data Types", SIGPLAN Notices, abril 1974.

LISK77 LISKOV, B. H., SNYDER, A., ATKINSON, R. y SHAFFERT, C., "Abstraction Mechanisms in CLU", CACM, Agosto de 1977.

LOCH77 LOCHOVSKY, F. H., AND TSICHRITZIS, D. C. 1977. User performance considerations in DBMS selection. In Proceedings of the International Conference on Management of Data (Toronto, Ontario, Aug. 3-5). ACM, New York, pp. 128-134.

LOPE85 LOPEZ, SANTIBAÑEZ S., "Herramienta de Software y Generadores de Programas para Elevar la Productividad del Programador.", II Congreso Internacional de Ingeniería en Comunicaciones Eléctricas y Electrónicas, AMICEE, México, D. F., Noviembre 1985, pp. 42-46.

MANN75 MANNA, Z., y R. WALDINGER, "Knowledge and Reasoning in Program Synthesis ", Artificial Intelligence, 6(2), 1975.

MANN78a MANNA Z., y R. WALDINGER, "The Synthesis of Structure-Changing Programs." In Proceedings of 3rd International Conference on Software Enginering (Atlanta Ga., May 10-12). IEEE, New York, pp. 175-187, 1978.

MANN78b MANNA Z., y R. WALDINGER, "DEDALUS-The Deductive Algorithm Ur-synthesizer." In Proceedings of National Computer Conference (Anaheim, Calif., JUne 5-8), vol. 47. AFIPS Press, Reston, Va., pp. 683- 690, 1978.

MANN79 MANNA Z., y R. WALDINGER, "Synthesis: Dreams =====> Programs". IEEE TRans. Softw. Eng. SE-5, 4, 294-328, 1979.

McCU77 McCUNE, B., "The PSI Program Model Builder: Synthesis of Very High Level Programs," Proceedings of the Symposium on Artificial Intelligence and Programming Languages, Agosto 1977.

McCU79 McCUNE, B. P., "Building Program Models Incrementally from Informal Descriptions". Ph.D. dissertation, STAN-CS-79-772, Computing Science Dept., Stanford Univ., Stanford, Calif., 1979.

MORI85 MORICONI M. y HARE, D. F., "Visualizing Program Designs Through PegaSys.", Computer, Vol. 18, Agosto 1985, pp. 72-85.

NASS73 NASSI, I. y SHNEIDERMAN, B., "Flowchart Techniques for Structured Programming", ACM Sigplan Notices, Vol. 8, No. 8, Agosto 1973, pp.12-26.

NEWM68 NEWMAN, W. M., "A Graphical Language for Display Programming.", Int'l Computer Graphics Symp., Brunel University, Uxbridge, England, Aug. 1968.

NEWM79 NEWMAN, W. M. y SPROULL, R. F., "Principles of Interactive Computer Graphics." (2 ed.) Mc. Graw-Hill, New York, 1979.

NUTT79 NUTT, G. J., Y ELLIS, C. A., "On the Equivalence of Office Models", Rep. SSL-79-8, Xerox Palo Alto Research Center, Palo Alto, Calif., Dec. 1979.

NUTT81 NUTT, G. j., y RICCI P. A., "Quinault: An Office Modeling System", IEEE Computer, Mayo 1981, pp. 41-57.

ORR77 ORR, KENNETH T., "Structured Systems Development", New York, Yourdon Press, 1975.

PANO79 PANOFSKY, ERWIN, "El significado en las artes visuales", Alianza Forma, Madrid, 1979.

PERL75 PERLIS, A., "Toward an APL Compiler", Research Report 24, Computer Science Department, Yale University, Marzo 1975.

POWE83 POWEL, M. L. y LINTON M. A., "Visual Abstraction in an Interactive Programming Environment.", Proc. Sigplan Symp. Programming Language Issues in Software Systems 83, publicado como Sigplan Notices, Vol 18, No. 6, Junio 1983, pp. 14-21.

PRYW77 PRYWES, N., "Automatic Generation of Computer Programs", in Computers, Volume 16, M. Rubinoff and M. Yovits (ed.) Academic Press, 1977.

PRYW79 PRYWES, N.S., PNUELI A., SHASTRY, S., "Use of a nonprocedural Specification Language and Associated Program Generator in Software Development". ACM Transactions Programming Languages Systems, Vol 1, No. 2. Octubre 1979, pp. 196-217.

PURC84 PURCELL, W. R., "Como Comprender las Finanzas de una Compañia (Un Enfoque Gráfico)." , Ed. Norma, Bogotá Colombia., 1984.

PURV83 PURVY, R. FARRELL, J. y KLOSE, P. "The Design of Star's Records Processing: Data Processing for the Non-computer Professional." ACM Trans. Office Information Systems, Vol 1., No. 1, Enero 1983, pp. 3-24.

RAMI73 RAMIREZ, J. A., "Automatic Generation of Data Conversion Programs Using a Data Description Language." Ph. D. Diss in Computer Science, Univ. de Pennsylvania, Philadelphia, Pa., 1973.

REIS83 REISS, S. P., "PECAN: Program Development Systems that Support Multiple Views", Tech. Report CS 83-29, Depto. de Ciencias de la Computación, Universidad de Brown, Providence, R. I., 1983.

REIS84 REISS, S. P., "Graphical Program Develoment with PECAN Program Develoment Systems." Proc. ACM Sigsoft-Sigplan Software Engineering Symp. Practical Software Develoment Enviroments, Abril 1984. Impresa como Sigplan Notices, Vol 19, No. 5, May 1984, pp. 30-41.

RIN76 RIN, N. A., "Automatic Generation of Business Data Processing Programs from a non-procedural Laguage". Ph.D. Diss in Computer and Information Science, Unv. de Pennsylvania, Philadelphia, 1976.

ROSS77 ROSS, D. T., "Structured Analysis: a Language for Comunication Ideas", IEEE Trans. Software Eng., 3, 1, pp. 16-33, 1977.

RUTH76 RUTH, G., "Protosystem I: An Automatic Programming System Prototype", LCS-TM-72, Laboratory for Computer Science, Massachusetts Institute of Technology, Julio 1976.

SAMM72 SAMMET, J.E., "An Overview of Programming Language for Specialized Application Areas," Proceding of 1972 Srping Joiant Computer Conference.

SCHM77 SCHMIDT, J. W. 1977. Some high-level languaje constructs for data of type relation. ACM Trans. Database Syst. 2,3 (Sep.), 247-261.

SCHM54 SCHMID, CALVIN F., "Handbook of Graphic Presentation", Ronald Press New York (1954).

SCHW73 SCHWARTZ, J., "On Programming: An Interim Report on the SETL Proyect", New York University, 1973.

SCOT88 SCOTT, DANFORTH y TOMLINSON, CHRIS., "Type Theories and Object-Oriented Programming". ACM Computing Surveys, Vol 20, No 1, Marzo 1988. 30-72.

SENK73 SENKO, M. E., ALTMAN, E. B., ASTRAHAN, M. M., y FEHDER, D. L., "Data Structures and accessing in Data-Base Systems", IBM System J., Vol. 12, pp. 30-93.

SHU75 SHU, NAN, HOUSEL, B. C. y LUM, V. Y., "CONVERT: A High Level Translation Defination Language for Data Conversion." CACM, Vol. 18, No. 10, Oct. 1975, pp. 557-567.

SHU82 SHU, NAN C., "Specification of Forms Processing and Business Procedures for Office Automation." IEEE Trans. Software Engineering, Vol SE-8, No. 5, Sep. 1982, pp. 499-512.

SHU85 SHU, NAN C., "FORMAL: A Forms-Oriented, Visual-Directed Application Development System.", Computer IEEE, Agosto 1985, pp. 38-49.

SMIT75 SMITH, D. C., "Pygmalion: A Creative Programming Enviroment", Disertación PhD, Depto. de Ciencias de la Computación, Universidad de Stanford (Tech. Report STAN-CS-75-499), 1975.

SMIT77 SMITH, MILLES J. Y SMITH D. C. P., "Database Abstractions: Aggregation and Generalization". ACM Trans Database Syst. 2, 2, (junio 1977), pp. 107-133.

SMIT81 SMITH, J. M., FOX, S., AND LANDERS, T. 1981. Reference manual for ADAPLEX. Tech. Rep. CCA-81-02, Computer Corporation of America, Cambridge, Mass.

SOFT76 SOFTECH, Inc. "TRAIDEX Need's and implementation Study (Final Report)". DARPA Contract *MDA 903-75-C-0224. Mayo 1976. (Disponible en: NTIS AD-A024 861).

STON76 STONEBRAKER, M., WONG, E., KREPS, P., AND HELD, G. 1976. The design and implementation of INGRES. ACM Trans. Database Syst. 1, 3 (Sept.), 189-222.

STON82 STONEBRAKER, M., y KALASH, J., "TIMBER: A Sophisticated Relation Browser.", Proc. Eighth VLDB, México City, Septiembre 1982, pp. 1-10.

SUTH63 SUTHERLAND, I. B., "Sketchpad, a Man-Machine Graphical Communication System.", Porc. AFIPS Conf., Vol. 23, 1963 SJCC, AFIPS Press, Reston, Va., 1963, pp. 329-346.

TEIT77 TEITELMAN, W., "A display oriented programmer's assistant", Proc. 5th Int. Jt. Conf. Artificial Intelligence, 1977, pp. 905-915

THAC79 THACKER, C. P., McCREIGHT, F. M., LAMPSON, B. W., SPROULL, R. F., y BOGGS, D. R., "ALTO: A personal computer". Computer Structures: Readings and examples, D. Siewiorck, C. G. Bell, y A. Newell (Eds.).

WARR76 WARNIER, JEAN-DOMINIQUE, "Logical Construction of Programs.", New York: Van Nostrand Reinhold, 1976.

WILS75 WILSON, M. L., "The Information Automat approach to design and implementation of computer-based systems.", IBM FSD Information Automat Tech Rpt., Junio 1975.

WULF76 WULF, W. A., LONDON, R. y SHAW, M., "An Introduction to the Construction and Verification of Alphard Programs", IEEE Transactions on Software Engineering, SE-2, pp. 253-264. 1976.

YOUR75 YOURDON, E. y CONSTANTINE, L. L., "Structured Design" Yourdon Inc., New York, 1975

ZLOO75 ZLOOF, M. M., "Query by Example: The Invocation and Definition of Tables and Forms.", Proc. First International Conference on Very Large Databases, Sep. 1975.

ZLOO77 ZLOOF, M. M.,"Query-by-Example: A Data Base Language." IBM Systems Journal 16, No. 4, 1977, 324-343.

ZLOO77a ZLOOF, M. y S. DE JONG, "The System for Business Automation: Programming Language," Communications of the ACM, 20 (6), Junio 1977.

ZLOO82 ZLOOF, M. M., "Office-by-Example: A Business Language that Unifies Data and World Processing and Electronic Mail." IBM Systems Journal 21, No. 3, 1982.

El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó esta tesis para obtener el grado de Doctor en Ciencias, el 14 de marzo de 1991.

Dra. Ana María Antonia Martínez Enríquez

Dr. Guillermo Benito Morales Luna

Dr. Jaime Rangel Mondragón

Dr. Adolfo Guzmán Arenas

Dr. José Luis Gordillo Moscoso