Universidad Nacional Autónoma de México

COLEGIO DE CIENCIAS Y HUMANIDADES
UNIDAD ACADÉMICA DE LOS CICLOS
PROFESIONALES Y POSGRADO

INSTITUTO DE MATEMATICAS APLICADAS Y SISTEMAS
MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

ESTRUCTURAS DE ALMACENAMIENTO PARA OBJETOS COMPLEJOS DE CAD

QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS DE LA COMPUTACION

Presenta

ROLANDO SAMUEL CARRERA SANCHEZ

OCTUBRE 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

PREFACIO

El uso de las computadoras como auxiliar, en casi todos los campos del saber humano, se ha vuelto práctica común. Sin embargo, es frecuente que el desarrollo de los sistemas computacionales, para cada campo, se realice en forma independiente. Tal tendencia sólo se invierte cuando, debido a la complejidad de las aplicaciones de un área, resulta necesario unir la experiencia que ha tenido otra área para resolver los problemas de la primera. Tal es el caso de los sistemas en los que la computadora se usa como auxiliar en el diseño (CAD) de maquinaria, aviones, barcos, circuitos electrónicos, plantas industriales y toda clase de artefactos y sistemas. En estos sistemas para CAD se ha encontrado la necesidad de manejar gran cantidad de información en forma eficiente, es decir se necesita mantener una base de datos de diseño, dentro de la cual se pueda tener acceso a información de piezas por medio de catálogos en linea, a los datos que describen el diseño los cuales se deben poder interrelacionar o asociar en forma compleja. Por otro lado se tiene el área del manejo de la información (datos), i.e. la que se ha dedicado al desarrollo de los sistemas manejadores de bases de datos (SMBD), que proveen el medio adecuado para el almacenamiento, clasificación, control y explotación de datos.

Desde sus inicios estas dos áreas, como muchas otras, se han desarrollado en forma independiente, pero en fecha reciente y debido a que en CAD, como en muchas otras áreas de la computación, se necesita manejar eficientemente la gran cantidad de datos que surgen de la complejidad de los problemas de diseño que se desean resolver, los investigadores han tenido que acudir al área de bases de datos en busca de ayuda para controlar su problema de manejo

de datos. De la unión de bases de datos con otros campos han nacido nuevas áreas hibridas con sus propios problemas particulares para la representación de datos. A continuación se listan algunas de ellas.

a. Bases de conocimiento. En los sistemas expertos la información puede ser representada a través de un conjunto de hechos, los cuales pueden ser vistos como una base de datos conteniendo el conocimiento del experto. Tales bases de datos son conocidas como bases de conocimiento. En los sistemas de bases de datos existentes no hay facilidades para almacenar este tipo de información, por lo que se están desarrollando los modelos y estructuras de datos adecuados para ese propósito.

b. Bases de datos para diseño. En los sistemas para CAD frecuentemente se encuentran datos que tienen estructuras internas complejas, tales datos generalmente son demasiado grandes, a diferencia de los encontrados en las aplicaciones convencionales como las bancarias, por lo que resulta necesario desarrollar nuevos modelos y estructuras de almacenamiento. En caso que los datos de un objeto de diseño no quepan en un sólo registro de disco, el sistema debe proveer el medio para almacenar esa informaccion en varios registros haciendo posible su posterior recuperación.

c. Bases de datos estadisticas. Estos sistemas requieren que su lenguaje de manipulación de datos tenga el mismo poder estadistico que proveen los paquetes de cálculo estadistico, así como un lenguaje de

definición de datos adecuado de manera que se permita asegurar que las preguntas que se hagan a la base de datos nunca permitan deducir a que persona, empresa o entidad pertenecen los resultados obtenidos. Esto significa que se asegura la confidenciabilidad de la base de datos estadística. Por otro lado los modelos de datos y estructuras que proporcionan los manejadores de bases de datos convencionales no son eficientes para guardar y recuperar eficientemente la información estadística ya que están orientados a almacenar juntos todos los atributos que corresponden a un renglón en las tablas, mientras que para realizar estadísticas lo que se necesita mantener junto, para reducir el tiempo de acceso, es la información referente a una columna de una tabla [CARR85].

d. Bases de datos para la automatización de oficinas o de otros ambientes de trabajo. Se están usando bases de datos como un componente altamente apreciado en sistemas desarrollados para automatizar el desarrollo de actividades en empresas y para simplificar la interacción del usuario con sistamas de cómputo complejos. La representación de los componentes de éstos ambientes en las bases de datos requiere la extensión de los modelos así como nuevas estructuras de almacenamiento que permitan mantener la duplicación de datos al mínimo.

Entre los principales problemas que todas estas áreas comparten, destaca la necesidad de contar con el sistema de almacenamiento y el modelo de datos adecuados para almacenar en forma eficiente y completa los datos de cada aplicación. La experiencia ha mostrado que tener un modelo de datos y

una estructura de almacenamiento comunes para todas las áreas no es una solución eficiente [KORH86, UMED85]. Por eso, resulta importante desarrollar técnicas adecuadas para cada caso de estudio, considerando así la trascendencia que el modelo y las estructuras de datos que se utilicen tienen sobre el comportamiento de cada sistema particular.

En el caso específico de CAD, para poder representar la información de una pieza o equipo que se está diseñando se requiere almacenar una cantidad enorme de datos. Además, se debe poder mantener las asociaciones (interrelaciones) que existen entre diferentes piezas, obligando a que el modelo que se use para representar el artefacto bajo diseño sea más complejo que aquellos usados por los manejadores de bases de datos comerciales.

En el IIMAS de la UNAM se ha estado desarrollando un sistema manejador de bases de datos para CAD (SMBD para CAD) [BUCA85] y [BUCA86b]. El propósito del trabajo que aquí se reporta es desarrollar el sistema de almacenamiento para dicho SMBD para CAD, el cual debe estar basado en un modelo de datos adecuado para almacenar en forma completa y eficiente los datos del diseño y las asociaciones lógicas y físicas que guardan entre sí. En el área de bases de datos se utilizan tres modelos de datos para representar la información de tipo administrativa del mundo de los negocios, a continuación se describe el uso de cada uno de estos modelos en la solución del problema de CAD.

a. El modelo jerárquico sólo permite almacenar, directamente, datos que estén organizados en forma jerárquica. Si se requiere representar una organización más compleja como lo es una red, se tendrá que recurrir a la repetición de datos con los problemas consecuentes para controlar la proliferación y actualización de los datos repetidos.

b. El modelo de redes (nos referimos al CODASYL [CODA71, TZID82, KORH86]) permite salvar el problema de repetición de datos que presenta el modelo jerárquico, sin embargo no permite diferenciar de que clase son las asociaciones (aristas) entre los vértices de la red; por lo tanto, es necesario extender este modelo o crear uno nuevo que contemple la posibilidad de manipular asociaciones de distintas clases como ocurre en el mundo real. Otro problema es que para mantener relaciones complejas de muchos a muchos se tiene que definir un nodo intermedio que sirva como puente de enlace.

c. El modelo de tablas relacionales tampoco permite, como parte del modelo, relacionar diferentes n-adas (o vértices en el caso de la red). Para lograr representar las asociaciones entre n-adas se tiene que definir una relación intermedia que sirva como enlace (esta solución nos recuerda la que se dió al mismo problema en el modelo de redes CODASYL).

En los tres modelos falta la forma de almacenar mayor información semántica acerca de los datos y de sus asociaciones [TZID82]. De hecho, la comunidad que estudia

las bases de datos lo ha reconocido y ha empezado a desarrollar otros modelos de mayor nivel de abstracción, e.g. el modelo entidad-relación que fue desarrollado como una herramienta teórica para poder representar modelos más reales del mundo durante la fase de diseño lógico de las bases de datos. Sin embargo debido a la carencia de SMBDs que esten basados en este modelo (i.e. hay muchos SMBDs basados en los tres modelos anteriores y uno o dos basados en el entidad-relación), al pasar a la etapa de diseño físico el diseñador tiene que traducir el diseño lógico a uno de los tres modelos anteriormente descritos.

Además se ha encontrado que el modelo entidad-relación presenta algunas deficiencias en su poder de modelado. Lo cual provocó que al enfrentarse con la falta de un modelo con mayor nivel de abstracción que pueda ser usado durante las etapas lógica y física del diseño, en el IIMAS se decidió usar un modelo de datos orientado a manejar objetos moleculares agregándole la capacidad de representar asociaciones entre los mismos. Adicionalmente, el modelo usado permite que las asociaciones entre los objetos formen redes, de asociaciones diferentes, y no sólo jerarquías, de una sola clase de asociación, solución que se da comúnmente en el área de CAD a este problema de representación debido a que generalmente se busca velocidad en la respuesta y no mayor poder de modelado del mundo real en el que las asociaciones en pocas ocasiones resultan ser jerárquicas o que sólo exista una clase de asociación entre los objetos.

Uno de los objetivos de esta solución es que en el sistema de almacenamiento se mantenga un balance adecuado entre la velocidad del sistema y el poder de representación

para modelar el mundo real.

Para realizar este sistema se necesitaba que otro subsistema del SMBD para CAD estuviera desarrollado, al menos en forma parcial, este es el Diccionario de Datos (DD) y se encarga de llevar el control de todas las definiciones estructurales de los objetos, i.e. de sus atributos, así como de las definiciones de las asociaciones válidas entre determinadas clases o instancias de objetos. Sin embargo debido a que el sistema no había sido desarrollado se tuvo que:

a. Definir las especificaciones del Diccionario de datos, tomando en cuenta las necesidades del módulo de almacenamiento principalmente; aunque también se tomaron algunas decisiones con respecto a otros módulos del SMBD para CAD, tales como el sistema de manejo de restricciones y el de plantillas y símbolos especiales.

b. Diseñar un simulador de las partes del DD (especificado en el apartado a) que hicieran posible crear clases de objetos, para probar que el sistema de almacemaniento pudiera manipular diferentes clases de objetos y sus instancias.

c. Del diseño del simulador mencionado en el punto anterior, se decidió omitir la parte que debía encargarse de describir la definición de las clases de asociaciones por no ser indispensable para el funcionamiento del sistema de almacenamiento. Aunque deberá ser el diccionario de datos, cuando esté

terminado, el que se encarque de verificar que la asociación entre dos clases o instancias de objetos sea válida i.e. que esté definida en el DD.

d. Programar el emulador del diccionario de datos descrito en los apartados b y c.

Una vez terminada esta parte del proyecto se dedicaron los esfuerzos al diseño y desarrollo del sistema de almacenamiento para objetos. Las características más sobresalientes del mismo son:

a. El modelo del sistema está basado en objetos complejos.

b. Permite representar redes de diferentes asociaciones entre los objetos.

c. Es capaz de manipular tanto atributos de longitud fija como variable.

d. Permite accesar los objetos directamente, capacidad que no se puede igualar fácilmente en los sistemas jerárquicos.

e. Evita almacenar repetidamente todos los objetos que se encuentran duplicados en trayectorias de diferentes asociaciones. Esto resulta difícil de evitarse en caso de utilizar sistemas jerárquicos con el consecuente problema de mantener las copias actualizadas.

Sin embargo, han quedado abiertos gran cantidad de problemas que aún se deben resolver. Varios de ellos se fundan en la falta de un marco teórico que respalde la idea de objetos como un modelo. Esto se debe a que el paradigma de objetos está basado en las ideas de un lenguaje de programación y no en un modelo de datos como el modelo relacional que tiene una teoría completa [CODE70] que lo respalda[1].

Se podría explorar la posibilidad de realizar y diseñar un lenguaje de consulta basado en el álgebra o en el cálculo como el caso del modelo relacional [ULLJ80]. Tal lenguaje deberá extenderse para resolver las desventajas que presenta el modelo relacional debido a que falla en la representación de asociaciones entre los objetos, la encapsulación de nuevas operaciones como parte de la definición de cada clase de objetos, la falta de poder del lenguaje para realizar algunas tareas como cálculos aritméticos o que requieran de ciclos para su realización y otros.

Este trabajo está enfocado a resolver el problema de bases de datos para CAD en áreas donde se requieren grandes bases de datos (más de diez mil objetos forman el diseño del sistema o artefacto), como es el caso del diseño de grandes plantas industriales; Sin embargo, sus resultados pueden aplicarse, en forma parcial, a otras áreas de CAD, de la inteligencia artificial, de interfases amigables,

---

1. Al estar basado en conjuntos el modelo relacional puede utilizar toda la teoría acerca de conjuntos que se ha desarrollado a lo largo de muchos años.

automatización de oficinas, etcétera.

Organización del trabajo

El reporte escrito de este trabajo está organizado de la siguiente manera: el capítulo 1 es la introducción donde se presentan la motivación y los objetivos del trabajo, así como una explicación global de lo que son los sistemas para CAD y la necesidad de crear manejadores de bases de datos especiales para este área: el segundo capítulo se encarga de presentar las características de lo que en el IIMAS se ha considerado debe ser un manejador de bases de datos para CAD. En el tercer capítulo se discuten los parámetro de diseño, el modelo de datos y las operaciones que deben soportar dos módulos del SMBD para CAD así como la realización parcial del primero, el diccionario de datos; El cuarto capítulo trata de la realización total del segundo, el sistema para el almacenamiento de los objetos complejos y sus asociaciones, que es el motivo principal de este trabajo. El primer módulo no formaba parte de este trabajo, pero por la incidencia directa que tiene sobre el módulo de almacenamiento se tuvo que realizar el diseño e implantación de una parte, con la que se necesitaba contar, que permite almacenar las descripciones de los objetos y de los atributos que los componen. El quinto capítulo presenta la discusión del sistema y las conclusiones, incluyendo un apartado en el que se tratan las etapas futuras. En este último se presenta el diseño de la segunda etapa del sistema, en el que se propone ya no utilizar el sistema de archivos de UNIX para realizar el acceso a disco, sino que se va ha hacer un sistema ya no de archivos sino de objetos que agilice el acceso al disco desde el punto de vista de una arquitectura basada en objetos. El diseño presentado ya

está listo para que se inicie otro trabajo de tesis.

CAPITULO 1
INTRODUCCIÓN

1.1 Motivación

Resulta indiscutible la importancia de realizar el diseño cuidadoso y perfectamente documentado de cualquier artefacto antes de pensar siquiera en lanzarse a la aventura de su fabricación. Esto es debido a que siempre existirán errores en las diversas etapas del diseño que se acumularán, causando retrasos en la fabricación, montaje y puesta en marcha del dispositivo. Tales retardos tendrían efectos negativos sobre la inversión realizada, por lo cual se hace necesario desarrollar, por un lado, programas de aplicación que ayuden a esta tarea y por otro lado desarrollar herramientas que permitan integrar el conjunto de tales aplicaciones a través de un solo sistema común que se encargue del control y manejo de los datos.

Debido al interés que en los últimos años han despertado algunas de las aplicaciones de la ingeniería de diseño como el diseño de circuitos de muy alta escala de integración (conocidos como VLSI[1]) [LORR83], de naves espaciales [BALR83] y de plantas de procesos, en las que se tienen que manejar miles de compuertas lógicas, de partes de las naves o de componentes de la plantas respectivamente, la comunidad de bases de datos comenzó a enfocar su atención en sistemas manejadores de bases de datos (SMBD) para CAD y se encontró con muchos problemas interesantes.

---

1 VLSI son las siglas para very large scale integrated circuits

Desde entonces se han hecho cantidad de contribuciones valiosas. Sin embargo, la situación que refleja la literatura es la siguiente: hay artículos en los que se establecen posturas y especificaciones de requirimientos, pero muestran muy poca información acerca de posibles soluciones o presentan soluciones específicas a subproblemas pequeños, la mayor parte de ellos inclinados hacia el diseño de VLSI. No obstante, aunque VLSI presenta problemas serios, existen otras disciplinas tal como el diseño de naves, automóviles, plantas de procesos químicos, y otras más en las que el uso de la computadora como auxiliar en el diseño presenta finos desafíos como lo son el desarrollar nuevas arquitecturas para sistemas manejadores de bases de datos; idear nuevos esquemas para el control de las transacciones de diseño; nuevos modelos de datos más adecuados, además de presentar requirimientos semejantes a los de CAD para VLSI.

En este sentido, el proyecto "integrated programs for aerospace-vehicle design" (IPAD) iniciado en 1976, establece un marco de referencia al determinar las caracteristicas que debe tener un ambiente adecuado para desarrollar sistemas integrales de CAD en un problema de diseño de gran magnitud. Como resultado del proyecto se concluyó que era necesario contar con un nuevo manejador de bases de datos en el que se resovieran las deficiencias que los SMBD comerciales presentan al ser usados para desarrollar aplicaciones de CAD[2]. Esta conclusión permanece vigente en nuestros dias

---

2. Más adelante se explican ampliamente cuales son tales deficiencias.

y se está trabajando en desarrollar esta nueva generación de SMBD.

Gracias a su gran confiabilidad y rapidez para realizar cálculos tediosos y repetitivos la computadora ha surgido como la herramienta que puede ayudar durante las diversas etapas del trabajo de diseño. Esto ha dado origen al desarrollo de técnicas para el uso de la computadora como auxiliar en el diseño (CAD)[3], mismas que han surgido como respuesta a la necesidad de maximizar la relación entre la inversión que se realiza y los resultados que se obtienen.

Las áreas que se beneficiaron principalmente con los resultados de todos estos esfuerzos de investigación y desarrollo en CAD han sido las áreas de electrónica (VLSI), diseño mecánico (IPAD) y desarrollo de diagramas o dibujos (VLSI e IPAD), lo cual trajo como consecuencia que su productividad se incrementara enormemente.

1.1.1 La aparición de CAD

Al igual que ha sucedido en otras áreas de la computación, al iniciarse el uso de la computadora en el área del diseño, los programas de computadora sólo emulaban los métodos tradicionales del cálculo manual. Esto se logró identificando los puntos de decisión importantes y desglosando los procedimientos de análisis y diseño manual

---

3. CAD también son las siglas en inglés de computer aided design.

en series de pasos sencillos, que al ser programados, dieron como resultado los primeros sistemas de CAD.

El ahorro en tiempo y costo que prosiguió al uso de estos sistemas demostró su factibilidad, por lo que CAD evolucionó rápidamente. Sin embargo, debido a diversas razones de índole económica e histórica, dicho desarrollo ha resultado en la proliferación de cientos de programas, para resolver problemas o subproblemas muy particulares de alguna aplicación de CAD, que han sido diseñados para funcionar aisladamente, por lo que al tratar de usar varios programas, que sólo resolvían un subproblema, para resolver un problema completo se crearon problemas de comunicación entre los programas obligando a buscar soluciones para lograr la integración exitosa de éstos en un solo sistema o paquete y eventualmente que se lograra el desarrollo de sistemas completos y modulares de propósito general.

Un ejemplo palpable de esta situación se presenta en el diseño de estructuras de edificios en el que se deben realizar varios tipos de cálculos como lo son el cálculo de fuerzas sísmicas, el cálculo de cargas estáticas y otros. La primera solución podría ser buscar diversos paquetes que resuelvan en forma independiente cada tipo de cálculo; sin embargo, esta solución no es deseable. La segunda solución que tendería más hacia lo que consideramos debe ser un sistema para CAD en estructuras incluiría los diversos tipos de cálculos para que el ingeniero civil no tuviera que preocuparse por conseguir un paquete distinto para cada clase de cálculo, evitando así que:

a. Tuviera que alimentar los datos a cada uno de los paquetes en su propio formato de entrada y

b. Que se tuviera que duplicar aquellos datos que fueran usados en ambos paquetes poniendo atención en alimentarlos en las unidades de medición que cada paquete usara i.e. uno de los paquetes podría hacer sus cálculos usando metros y el otro usando pies o centimetros.

Por otro lado, la evolución de CAD, como de todos los sistemas de "software" siempre ha sido muy dependiente de las tecnologías disponibles en el área de la computación. Con respecto a esto a continuación se discuten los principales inventos tecnológicos que han causado el mayor impacto en el desarrollo de CAD, ya sea porque hagan que algo sea posible, más sencillo, o más rápido

- El incremento en el poder y velocidad de las computadoras ha permitido el desarrollo exitoso de aplicaciones de ingeniería más grandes y complejas.

- La aparición de las unidades masivas para almacenamiento (discos magnéticos) han servido como el medio en el que se pueden mantener en línea los datos del proyecto de diseño y las bibliotecas de componentes de ingeniería, sus propiedades físicas y costos de los mismos. Esto ha permitido que estos datos puedan ser almacenados y consultados directamente por los programas de diseño o análisis.

- El desarrollo de periféricos para la captura de datos y el despliegue de información gráfica ha servido para que los programas de aplicación emitan la visualización completa de las diversas etapas del diseño al producir gráficas y planos del diseño junto con la tradicional salida en forma de datos aislados o tablas.

- Los procedimientos de diseño y de análisis han sido examinados y reestructurados desarrollándose técnicas propias para CAD en vez de sólo computarizar los métodos manuales. Así han aparecido sistemas que no solamente sirven para realizar el análisis del diseño, sino que además proveen facilidades tales como rutinas generales para facilitar la entrada y salida de los datos; para llevar el control de las diversas etapas del proyecto, de sus modificaciones, de su ruta crítica, de compras de materiales, de gastos, etcétera; acceso a bibliotecas de componentes, planos, diagramas y modelado.

1.1.2 CAD para plantas de procesos industriales

El propósito principal de los sistemas CAD para plantas de procesos es analizar las condiciones de diseño y construcción de dichas plantas de manera que conlleven el utilizar menos capital al mismo tiempo que se obtenga mayor eficiencia en la operación; así como mejorar la productividad de los despachos de ingeniería de diseño, alcanzando las soluciones más adecuadas en tiempos más cortos. Las principales herramientas de CAD para plantas de procesos [LEEM82] son: la ingeniería de procesos, las bases de datos, graficación, periféricos para computadoras, interfases amigables al usuario, administración y costos

entre otras.

Tabla 1-1. Etapas del desarrollo de pantas de proceso.

Estudios de factibilidad

Síntesis del proceso
- escoger rutas del proceso
- desarrollo de las hojas de flujo

Diseño del proceso
- condiciones de operación
- diseño funcional de las unidades

Diseño de los equipos

Diseño de la disposición de la planta y de las tuberías

Diseño del sistema de control e instrumentación

Estudios de operabilidad
- arranque, paro de la planta, cambio de régimen
- operación normal
- análisis de problemas y riesgos

Construcción de la planta
- control de existencias y adquisición de materiales
- fabricación (equipos)
- edificación

El diseño de plantas químicas y de procesos, al igual que en la mayoría de las aplicaciones, es complejo y está formado por una serie de actividades que deben efectuarse en paralelo para minimizar tiempos y costos. Estas tareas de diseño, que se muestran en la tabla 1-1, se han agrupado en tres fases: la ingeniería de procesos, la ingeniería de

detalle y la construcción de los equipos y edificios. Además, cada operación o movimiento dentro de estas tres fases debe estar estrechamente monitoreado por él o los administradores del proyecto.

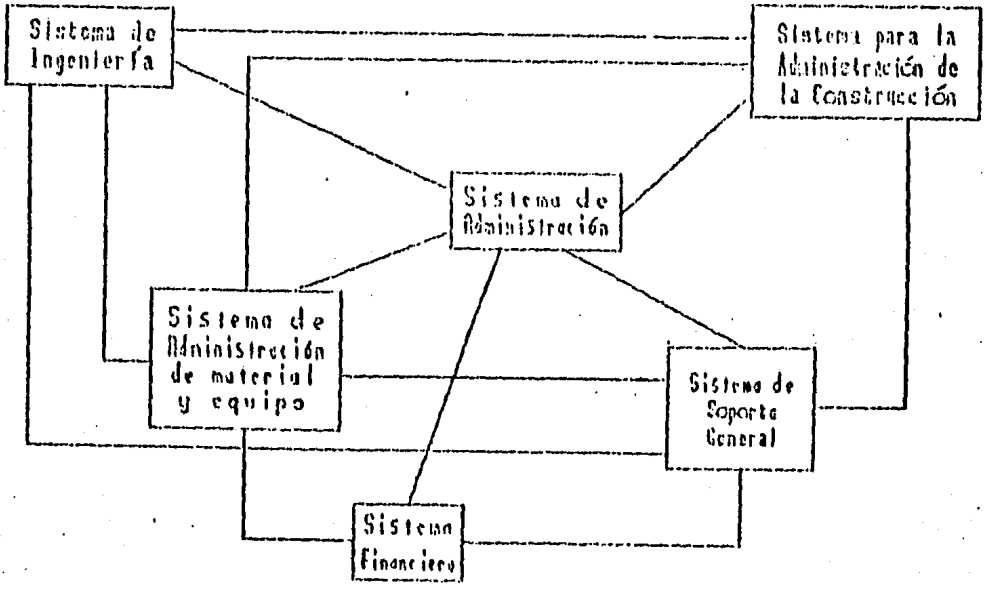


Figura 1-1. Las partes de un sistema CAE.

Esta complejidad ha sido, en parte, la causa de que en un principio los diseñadores de plantas de procesos no hayan tenido confianza en el uso de los sistemas de CAD, excepto en el desarrollo de hojas de flujo y para ayudar en la administración del proyecto. Estas dos actividades son mínimas en comparación con las otras que se tienen que realizar para el desarrollo de una planta (véase tabla 1-1).

Es claro el pobre uso que se le daba a CAD en este área. Otra de las causas de este problema ha sido la naturaleza conservadora de la industria, actitud que no tiene bases reales y se debe fundamentalmente a la falta de entendimiento de lo que son estas herramientas básicas, de cómo pueden ser usadas y de cúal es el posible impacto que pueden tener para contribuir a un diseño exitoso (en cuanto a rapidez, calidad y costos) de las plantas de proceso. Actualmente, esta actitud ha cambiado, principalmente por la necesidad que tienen los despachos de diseño de mantenerse competitivos dentro del ámbito de la industria, Esto hace que cada vez se usen más los sistemas de CAD, por lo que se actualmente se requieren sistemas más complejos y completos.

No hay limites rigurosos que indiquen que el uso de los sistemas de CAD sólo debe aplicarse a ciertas etapas del diseño, por lo que estos pueden crecer hasta formar lo que se conoce como sistema para ingeniería auxiliado por computadora (CAE)[4]. Este está formado por varias partes o subsistemas que pueden interactuar o ser independientes (véase la figura 1-1), cuyas funciones individuales se presentan en la tabla 1-1.

---

4. CAE son las siglas en inglés de computer aided engineering.

Tabla 1-2. Las Funciones de los subsistemas de CAE.

Sistema	Función
Ingeniería	Balances de energía y materiales Diseño de los equipos para procesos Diseño de los equipos mecánicos Generación de planos Generación de la especificación
Administración de los equipos y materiales	Ordenes de compra Control de despachos de órdenes Control de almacenes Facturación de Materiales
Administración de la construcción	Monitoreo de la construcción Análisis de productividad
Administración del proyecto	Estado del proyecto Planeación y predicción de recursos humanos Datos del personal
Financiero	Reportes de costos o gastos Predicción de gastos Contabilidad Presupuestos
Soporte general	Sistemas para oficina Estimación de costos Calendarización Bibliotecas de rutinas Recuperación de información

1.1.3 Las deficiencias de los sistemas actuales

El diseño de plantas de procesos requiere la interacción de muchas disciplinas diferentes y de grupos de diseñadores que se especializan en áreas distintas. Esto provoca, entre otras cosas, tener que cubrir las necesidades diferentes de cada uno de los grupos del proyecto, así como el resolver problemas de comunicación entre estos grupos y los diferentes paquetes para permitir que se realicen varias

actividades en paralelo, reduciendo con ello tiempos y costos.

Al inicio los sistemas de CAD sólo se enfocaban a resolver una parte del diseño (para la administración del proyecto o para hojas de flujo del proceso), pero conforme fueron más aceptados, aparecieron nuevos paquetes orientados a apoyar otras partes del diseño (simulación, diseño de equipos, planos, estudios de factibilidad, de costos, etcétera). Esta mayor aceptación originó varios problemas: el primero fue de comunicación, i.e. cuando se requiere integrar varios paquetes que resuelven problemas parciales para la solción de un problema completo; el segundo es el de concurrencia, cuyo origen es la necesidad de que los datos sean accesados y compartidos por los diferentes equipos de diseño; en tercer término está la necesidad de mecanismos para recuperarse en caso de falla del sistema; y por último, enunciaremos la ausencia de interfases amigables hacia los usuarios que los ayuden a llevar el registro, control y acceso a sus datos.

1.2 Un sistema integral para CAD

Existen dos caminos para poder unir diferentes aplicaciones, independientes, de CAD para formar un sistema integral.

A. Integrar los paquetes existentes creando rutinas para reformatear los archivos entre cada pareja de aplicaciones.

B. Integrar los paquetes alrededor de un conjunto común de datos los que sólo pueden ser accesados a través de un manejador de bases de datos.

1.2.1 La integración de sistemas CAD sin usar bases de datos

Esta opción requiere del desarrollo de una rutina traductora para cada pareja de aplicaciones que tengan que comunicarse, de manera tal que sirva como interfase entre ellas al reformatear el archivo de salida de una aplicación para ser usado, con el nuevo formato, como entrada de la otra. Para que esta solución sea flexible y tenga éxito, es necesario que exista un programa traductor entre cada pareja que necesite comunicarse. Las desventajas de esta solución surgen del almacenamiento de datos redundantes entre cada punto de interfase. Estos datos redundantes, a menudo, tienen valores inconsistentes debido a fallas en el control de la propagación de actualizaciones, y tales inconsistencias poco a poco se van propagando a todo el sistema al usar los datos falseados en diversas etapas.

Los resultados producidos con el uso de esta solución han sido confusos debido a que al integrar paquetes independientes que trabajaban bien, no funcionan igual debido principalmente a incompatibilidades en los datos. Estos sistemas sólo tienen éxito cuando el número de interfases de traducción es pequeño y se lleva un control estricto para la propagación de cada cambio que se haga a los datos.

1.2.2 La integración de sistemas CAD usando bases de datos

La opción de realizar la integración utilizando un sistema manejador de bases de datos (SMBD) es más ambiciosa, pero al mismo tiempo también es más prometedora. Conforme los sistemas para CAD han crecido, se han topado con el problema de manipular sus datos y metadatos. Dado que existe un área de la computación que se ha especializado en el manejo de datos esta es la instancia más adecuada a la cual acudir en busca de ayuda. Ya que aunque es cierto que los sistemas para CAD han evolucionado, persisten otros problemas como:

a. Tener interfases amigables;

b. Mejores esquemas de control de concurrencia;

c. Mecanismos para recuperación contra fallas;

d. Sistemas que ayuden en el control y descripción de cientos o miles de datos de diferentes clases que se requieren para describir un sólo artefacto bajo diseño;

Los resultados producidos con el uso de esta solución han sido confusos debido a que al integrar paquetes independientes que trabajaban bien, no funcionan igual debido principalmente a incompatibilidades en los datos. Estos sistemas sólo tienen éxito cuando el número de interfases de traducción es pequeño y se lleva un control estricto para la propagación de cada cambio que se haga a los datos.

1.2.2 La integración de sistemas CAD usando bases de datos

La opción de realizar la integración utilizando un sistema manejador de bases de datos (SMBD) es más ambiciosa, pero al mismo tiempo también es más prometedora. Conforme los sistemas para CAD han crecido, se han topado con el problema de manipular sus datos y metadatos. Dado que existe un área de la computación que se ha especializado en el manejo de datos esta es la instancia más adecuada a la cual acudir en busca de ayuda. Ya que aunque es cierto que los sistemas para CAD han evolucionado, persisten otros problemas como:

a. Tener interfases amigables;

b. Mejores esquemas de control de concurrencia;

c. Mecanismos para recuperación contra fallas;

d. Sistemas que ayuden en el control y descripción de cientos o miles de datos de diferentes clases que se requieren para describir un sólo artefacto bajo diseño;

e. Disminuir o eliminar la duplicidad de datos cuando éstos fueran necesitados por distintos paquetes; y

f. Mantener consistentes y actualizados todos esos datos, estuvieran o no duplicados, de acuerdo a los cambios que los diseñadores vayan realizando durante el diseño.

Esta solución nos permite modelar en la base de datos cualquier artefacto que se encuentre bajo el proceso de diseño, representándolo por medio de un conjunto de datos interrelacionados. Las rutinas para inserción, supresión, actualización y recuperación que provee el SMBD funcionarán como el conducto único a través del cual las diversas aplicaciones podrán accesar los datos.

Algunas de las ventajas de esta solución son que tanto los programadores como los usuarios del sistema no deben preocuparse por llevar el registro y control de la localización física de los datos ni de los archivos del diseño, ni tampoco por el manejo de la concurrencia, recuperación contra fallas y demás mecanismos que ya son proporcionados por el SMBD; esto es, los usuarios pueden estar seguros de que siempre tendrán acceso a datos actualizados. Finalmente esta solución tiene la propiedad de facilitar el resolver nuevos requerimientos o integrar nuevas aplicaciones, fortaleciendo de esta manera la flexiblidad del sistema para facilitar el mantenimiento de las aplicaciones.

1.2.3 Deficiencias del uso de los manejadores convencionales en CAD

Los principales problemas encontrados al usar los SMBD convencionales para desarrollar las aplicaciones de CAD, fueron las restricciones que imponen tales sistemas sistemas en el sentido de que sólo manejan datos de tipos alfabético y numérico. Estas restricciones no causan mayores problemas para la representación de los datos que se manejan en el medio de las aplicaciones comerciales debido a que aquellos son principalmente de tipo alfanumérico y a que los artículos descritos comúnmente son muy sencillos. Pero la capacidad de estos sistemas para manejar la información de CAD es limitada ya que en la ingeniería de diseño los datos frecuentemente son pictográficos y los artículos que se deben describir en la base de datos resultan bastante complejos, los cuales requerirían cientos de campos distribuidos a lo largo de diversos tipos de registros para describirlos apropiadamente. Adicionalmente, la interdependencia de los datos impone nuevos problemas, debido a que se requiere que los cambios se propaguen automáticamente a través del diseño; finalmente se requieren mecanismos para el control de concurrencia de las transacciones largas de CAD, muy diferentes de los mecanismos de concurrencia de las transacciones cortas[5] de tipo administrativo.

---

5. Son las transacciones de poca duración en tiempo de proceso.

A pesar de estos problemas, el uso de los SMBD comerciales ha traido como consecuencia la reducción de las inconsistencias entre datos de diferentes aplicaciones ya que proporciona los mecanismos de consulta a la BD y de recuperación contra fallas; también se puede hacer uso del control de concurrencia de los SMBD comerciales mientras se desarrollan otros mecanismo más adecuado para manejar la concurrencia de las transacciones largas de CAD asi como los mecanismos para almacenar los datos de los objetos de diseño.

1.2.4 El problema de modelos de datos para CAD

Para facilitar la interacción entre los usuarios y las máquinas los sistemas para diseño deberían representar y manipular los objetos de la vida real en una manera similar a la normalmente utilizan los usuarios. Estos objetos a menudo son complejos y estan conformados por otros objetos; por ejemplo, en la ingeniería de diseño, los objetos consisten de decenas y hasta centenas de campos, dependiendo del grado de detalle con el que se trabaje. Si esta información es forzada dentro de un modelo plano como lo son el jerárquico, el reticular y el relacional, que carecen de poder semántico para representar las asociaciones complejas que existen entre objetos o entre éstos y sus componentes, tendrán que ser el usuario y/o el programa de aplicación quienes tengan que gastar tiempo y esfuerzo valioso en interpretar el modelo, con el riezgo de caer en alguna malinterpretación, debido a problemas en la representación semántica e ineficiencia en el procesamiento realizado por

el SMBD. Por lo tanto, es necesario contar con un modelo de objetos que permita una mejor representación de la realidad y que el ususario no deba preocuparse de las n-adas o de los registros sino de objetos.

Para darse cuenta del problema que representa almacenar y recuperar esta información en un modelo de datos como el relacional o el de redes, baste imaginarse el proceso de tratar de accesar decenas de diferentes tipos de registros y seleccionar de cada uno de ellos solamente los campos que se refieren al objeto que se desea "visualizar", unido esto al problema de determinar cuáles son los registros que almacenan alguna información de dicho objeto. En un modelo de datos basado en objetos se debe tener entonces la posibilidad de describir la composición de un objeto agrupando otros, así como el poder de describir la semántica bajo la que se hizo dicho agrupamiento, en este tipo de sistemas esto se ha logrado al permitir el uso de diferentes clases de asociaciones entre los objetos. Además el SMBD-CAD debe realizar las operaciones de actualización en forma correcta sin importar cúal objeto esté actualizando, y propagar dicha actualización hacia los demás usuarios que esten usando copias del objeto modificado. También debe permitir la herencia de atributos de un objeto hacia sus subobjetos, así como el cálculo de cualquier atributo de un objeto en función de los atributos de los subobjetos que lo formen.

1.2.5 Algunos sistemas de CAD que usan bases de datos

La mayor parte de los trabajos relacionados con el uso de manejadores de bases de datos problemas de diseño de proyectos de ingeniería son demasiado recientes para haber sido diseminados. Además, la mayor parte se han realizado en la industria por lo que su divulgación ha sido restringida debido al secreto industrial.

EL cambio de los sistemas de CAD independientes hacia sistemas totalmente integrados está todavía en su infancia [BALR83]. En la industria tales sistemas, por lo general, se realizan siguiendo metas de corto alcance ya que los costos tienen que ser justificados al corto plazo. Como ejemplo se tienen las bases de datos para proyectos de ingeniería de las compañías DuPont y Monsanto [BUCA80]. Aunque no existen detalles disponibles de estos sistemas, respecto al de Monsanto parece ser que la mayor parte del trabajo de diseño de la base de datos ha sido empírico, usa un SMBD jerárquico de la compañía IBM llamado IMS, y la mayor parte de sus bases de datos de diseño son específicas para cada aplicación limitando con esto la posibilidad de comunicación entre aplicaciones. DuPont hace uso de ADABAS e intenta extender su sistema de bases de datos para que funcione como la base de datos para el control del Proyecto global de diseño, sin embargo no hay mayores datos del estado de estos esfuerzos [DAWF82]. Es importante señalar que a pesar que se deben estar realizando otros sistemas, la competencia entre las compañías de ingeniería ha impedido la comunicación, la cooperación y la publicación de los enfoques y resultados obtenidos en esos esfuerzos.

1.3 Nuestro sistema de bases de datos para CAD

El SMBD para CAD propuesto por investigadores del IIMAS (cuya justificación y descripción puede ser encontrada en [BUCA85, BUCA86a y BUCA86b]) difiere de los sistemas convencionales en dos puntos principales: en primer lugar se trata de un sistema manejador de bases de datos especificamente orientado para realizar transacciones de diseño que pueden tardar en proceso desde unos segundos hasta semanas, lo cual es resultado directo del tiempo que se necesita para completar un diseño o un cambio al mismo, mientras que las transacciones convencionales se realizan en periodos muy cortos; y en segundo lugar, se tiene que al usar un modelo basado en objetos con relaciones para formar los agregados moleculares [BATD84], se posee la capacidad de modelar mayor cantidad de datos acerca del articulo bajo diseño, incluyendo la información de los papeles que desempeñan los objetos al formar parte de alguna relación entre ellos mismos, tal como generalizaciones (IS-A) o agregación (PART-OF, CONECTED-TO).

1.3.1 Caracteristicas de un SMBD para CAD

El programador de aplicaciones de CAD debe contar con un SMBD especial para CAD (que resuelva la mayor parte de las deficiencias que implica el uso de SMBD convencionales), que proporcione el ambiente adecuado para desarrollar sus programas evitando que él tenga que preocuparse en desarrollar además de su aplicación: interfases para hacer consultas, mecanismos de recuperación contra fallas, control de concurrencia para transacciones largas, reducir inconsistencias propagando las actualizaciones, llevar el

control de versiones del diseño y asegurar la consistencia de los datos con base en una serie de restricciones que se definan para llevar a cabo el diseño. Estos conceptos se amplían a continuación.

1.3.1.1 Manejo de diversos tipos de datos. Hay varios tipos de datos que los SMBD comerciales no pueden manejar. En especial se hace referencia a diversos tipos de datos de longitud variable que se encuentran frecuentemente en el manejo de diagramas del diseño en los que se utilizan cadenas de caracteres de longitud variable para la sección de notas del diagrama en la que se estipulan observaciones acerca del diseño; existe también la necesidad de manejar datos pictográficos, los cuales generalmente se almacenan en forma secuencial para agilizar el proceso de despliegue en la terminal gráfica.

1.3.1.2 Control de concurrencia para transacciones largas. En general el ingeniero de diseño trabaja durante horas o hasta días en forma interactiva antes de terminar una modificación a su diseño y decidir que está correcta. Esto significa que durante todo ese tiempo la base de datos estuvo procesando una sola transacción y, por lo tanto, durante ese lapso los datos pueden haber permanecido inconsistentes. Es precisamente esta duración tan larga de las transacciones de CAD lo que obliga a tener un esquema especial de control de concurrencia. En este caso se logra al separar las bases de datos de las áreas de trabajo (BD/AT) de cada diseñador y la base de datos del área central (BD/P) donde descansa la versión completa y consistente del diseño del proyecto, proporcionando

operaciones especiales para extraer parte de un diseño de la BD/P y llevarla a una BD/AT, así como para reintegrar el diseño modificado desde cualquier BD/AT hacia la BD/P. La operación de reintegración se debe hacer con base en la bitácora que lleva cada sistema manejador de las áreas de trabajo.

1.3.1.3 Propagación de actualizaciones. Con base en el mecanismo de control de concurrencia antes descrito, en el que los cambios de cada BD/AT son reintegrados uno a la vez hacia la BD/P, la propagación de las actualizaciones tiene que resolverse durante la reintegración de la copia verificada hacia la BD/P. Es en ese momento cuando se hace necesario saber qué diseñadores serán afectados por dichos cambios, de forma que se les notifique tal situación. Cuando algún cambio necesite ser autorizado por alguien más, no entrará en efecto hasta que dicha persona dé su autorización.

1.3.1.4 Control de versiones. Los cambios a la BD/P se hacen a través de revisiones las cuales son cambios bien documentados, para cada uno de los cuales se almacena el autor; la fecha del cambio, el nombre del que lo revisó y la fecha, así como el nombre del responsable de aprobar la modificación y la fecha. Con toda la información antes mencionada es posible recuperar en cualquier momento la versión que tenía el proyecto en una fecha determinada.

1.3.1.5 Verificador de restricciones y excepciones. La comprobación de restricciones y sus excepciones es el mecanismo a través del cual se propone asegurar que todos los datos del diseño cumplan con las especificaciones de funcionamiento, esto significa que operan dentro del rango de seguridad para el que fueron fabricados; por ejemplo, una tubería que por fabricación soporta una presión máxima determinada. Como la anterior, se hacen otras verificaciones más pero sin llegar a hacer tan complejo el manejador de restricciones que se pudiera pensar que se está tratando de entrar en el área de sistemas expertos. La diferencia principal estriba en que en estos últimos se manejan reglas y procesos de inferencia para sugerir posibles alternativas de diseño, en cambio en nuestro sistema sólo se hace la verificación de las restricciones y excepciones definidas por el usuario quién es el que se encarga de decidir las posibles opciones de diseño.

1.3.2 Organización de este SMBD para CAD

En general, el diseño de un proyecto se reparte entre varios diseñadores que se especializan en alguna de las áreas de diseño. Cada uno de dichos diseñadores trabaja independientemente en su propia estación de trabajo, en la cual únicamente se almacena la información que corresponde a su sección de entre todo el proyecto global de diseño. A este tipo de información se le ha denominado la Base de Datos del Area de Trabajo (BD/AT), y es en esta información sobre la que se llevan a cabo las transacciones de diseño (inserciones, modificaciones, etc.) en forma interactiva.

Junto con estas BD/ATs se necesita mantener la Base de Datos global del Proyecto o BD/P, en la cual se lleva la consolidación de todas las transacciones realizadas en las BD de las ATs, verificando la consistencia entre lo que ya está diseñado, de acuerdo a las especificaciones (restricciones) del diseño, y las modificaciones y adiciones que se han hecho en las diferentes BD/ATs.

Por otro lado, se necesitan otras bases de datos que servirán para realizar funciones de apoyo, pero que por su importancia dentro de nuestro ambiente del SMBD orientado a CAD deben ser mencionadas en forma explícita para poder entender plenamente los requerimientos de dicho sistema. Dichas BD de apoyo serán:

- La base de datos del catálogo de los objetos. Inicialmente contendrá la información de los objetos de uso común que se requieren en el diseño (tal como se encuentra en los catálogos de la industria). Con el tiempo en ella se irá almacenando la información estándar de los nuevos tipos o clases de objetos que se vayan dando de alta para extender el catálogo.

- La base de datos de las plantillas y símbolos particulares. Aquí se almacenará la información de los formatos particulares que se usarán para cada uno de los proyectos de los diferentes clientes, así como los símbolos estandarizados para generar dibujos.

- El diccionario de datos e información de control. En él se almacenarán todos los metadatos de la BD, los cuales incluyen la descripción de cada clase de objeto y la información que especificará la trayectoria de acceso que se usará para determinar la forma de hacer el agrupamiento eficiente de los datos en las BD/AT.

- La base de datos de las restricciones. Aquí se almacenará la información para realizar la verificación de restricciones y excepciones, las cuales pueden ser aplicables a alguna instancia particular de objeto o a todas las instancias de alguna clase de objeto.

Las transacciones largas de CAD se pueden manejar gracias a que se decidió proveer de espacios de trabajo separados para cada diseñador, por lo que se deberán soportar: la operación de extracción que permite transferir datos desde la BD/P hacia la BD/AT; la operación de reintegración, la cual transfiere las modificaciones, inserciones o supresiones hechas en la BD/AT hacia la BD/P realizando la verificación de restricciones y excepciones. El inicio y fin de cada transacción de CAD son representados por una operación de extracción y una de reintegración exitosa.

1.4 Objetivos del trabajo

Este trabajo sigue la tendencia de desarrollar herramientas que permitan mayor eficiencia y versatilidad en el área de CAD. Específicamente nos proponemos realizar un prototipo lógico y físico de las estructuras de datos para almacenar los objetos complejos, sus atributos, sus objetos componentes, así como las asociaciones que entre éstos pudieran existir. Nuestro interés es poner atención especial en que las asociaciones entre los objetos puedan ser manejadas como redes de diferentes clases de asociaciones y no sólo como jerarquías de una sola clase de asociación, que es la solución que se da comúnmente a este problema de representación.

Esto es porque generalmente se busca velocidad en la respuesta y no mayor poder en el proceso de modelado del mundo real, en el que generalmente las asociaciones resultan ser reticulares y pocas veces se da el caso en el que sean jerárquicas o que sólo exista una clase de asociación entre los objetos. Ante tal disyuntiva, en este sistema se busca obtener un balance entre su velocidad de respuesta y su capacidad de representación, balance en el que estamos dispuestos a perder una poca de velocidad si ganamos en el poder de representatividad del sistema con el propósito de poder modelar más adecuadamente las situaciones de diseño en las que se tengan redes de asociaciones.

Dos situaciones en las que se necesita modelar redes se encuentran en los grandes problemas de diseño de plantas químicas y en pequeños problemas de automatización de oficinas.

Para ejemplificar el primer caso expondremos una situación común de la ingeniería de procesos en la que una planta de procesos industriales puede requerir de varios procesos para la producción de un producto, v.gr. cuando se esta diseñando una planta de procesos industriales se tienen diversos procesos que estan siendo diseñados por diferentes grupos de diseño entonces se da el caso en que el mismo equipo o maquinaria está siendo utilizado por dos o más procesos, y si dentro de la base de diseño nosotros queremos evitar duplicados tenemos que hacer posible el manejo de redes o atenernos a que la otra solución sería duplicar los equipos que se comparten, junto con los duplicados de todas las piezas que constituyen a esos equipos duplicados, con los consecuentes problemas de mantener las copias actualizadas cada vez que uno de los grupos de diseño hiciera alguna modificación sobre el equipo común.

En el segundo caso expondremos el problema de manejar las compras de una empresa en la que cada departamento formula una requisición interna de compra y se la entrega a la oficina de compras, esta a su vez necesita reutilizar los datos que vienen en la forma de requisición para formular uno o varios pedidos (posiblemente a diferentes proveedores). Este es otro ejemplo claro en el que las redes evitarían duplicados ya que tanto la solicitud de compra como el pedido podrían compartir las descripciones y cantidades de los artículos que se desean adquirir.

En el Instituto de Matemáticas Aplicadas de la U.N.A.M. se propuso un proyecto para desarrollar un sistema manejador de bases de datos adecuado para aplicaciones en al área del diseño auxiliado por computadora (SMBD para CAD). Estas aplicaciones se caracterizan por requerir del manejo

simultaneo de cantidades muy grandes de información (10K elementos o más), con lo que se cruza el umbral hacia las grandes bases de datos. El trabajo que aquí se presenta surgió con el propósito de realizar un prototipo del subsistema de almacenamiento del SMBD para CAD propuesto en el IIMAS. Las carácteristicas sobresalientes de este subsistema son las siguientes:

1. Debe soportar el almacenamiento de objetos complejos.

2. Permitir representar jerarquias y/o redes de asociaciones entre ellos.

3. Y que los atributos de los objetos puedan ser de longitud variable o fija.

Para realizar el sistema se hizo un estudio para determinar las caracteristicas que debia tener el ambiente sería utilizado primero para el desarrollo y despúes para el uso o explotación del sistema. Las caracteristicas más importantes que se debian cumplir eran que: se necesitaba un ambiente versatil y poderoso que resultara adecuado para desarrollo de nuevos sistemas y que a la vez fuera standar en el medio del diseño auxiliado por computadora.

Estas son las razones por las que se decidió que el sistema se haría utilizando el sistema operativo UNIX con el propósito de estar a la par con los principales sistemas y estaciones de trabajo para CAD, ya que la mayoria usan UNIX. El lenguaje de desarrollo es "C" lo cual resulta lo más lógico si se está usando UNIX. Adicionalmente a las razones anteriores, UNIX provee un ambiente adecuado para el desarrollo de nuevos sistemas gracias al conjunto de

herramientas para el desarrollo de programas que forman parte del mismo, tales como "make" que sirve para controlar la compilación modular de un sistema llevando el control de las dependencias entre los módulos y "SCCS" (Siglas para source control code system) que permite llevar el control de diferentes versiones de los programas.

Por otro lado, hay que resaltar que anteriormente ya se habían comenzado a desarrollar, en el IIMAS, algunas otras herramientas para CAD que también fueron desarolladas en UNIX y dado que siempre es deseable mantener la compatibilidad con lo desarrollado, esto influyó a que se decidiera usar UNIX en nuestro trabajo.

CAPITULO 2
ORGANIZACION DEL ALMACENAMIENTO

Diferentes sistemas de CAD han utilizado varios esquemas para el almacenamiento de datos en disco. En este capitulo presentamos la organización del almacenamiento de los datos a través de dos alternativas globales: la primera trata del almacenamiento sin el uso de sistemas manejadores de bases de datos (SMBD); y la segunda se refiere al caso cuando si se han usado estos sistemas. En el último caso, nuestro análisis toma en cuenta si el SMBD está o no diseñado para soportar el manejo de datos de CAD.

A través de las diversas secciones también se analiza el problema de la repetición del código fuente de las operaciones sobre los datos y se expone la necesidad de que estas sean manejadas a través de algún tipo de mecanismo centralizador, para evitar la necesidad de actualizar decenas de copias, del código fuente, de una operación cuando se realice algún cambio en la estructura de los datos o en la forma de realizar la operación.

2.1 Almacenamiento en sistemas de CAD sin usar SMBDs

Es característico de los sistemas de CAD, que manejan el almacenamiento sin utilizar SMBDs, realizar el almacenamiento través de archivos separados [BUCA86b], donde la organización interna de cada archivo frecuentemente es diferente ya que cada programa de aplicación es responsable de manejar el almacenamiento de sus datos, y por lo tanto, también lo es de mantener sus propias estructuras de datos para los archivos. Como resultado de esta situación se presentan varios problemas al tratar de integrar un conjunto de paquetes separados para obtener un sistema de CAD integral que proporcione mayor diversidad de opciones que puedan ser usadas durante un diseño.

El primer problema se origina en la necesidad de integrar sistemas de aplicaciones independientes. En la figura 2-1 mostramos la complejidad que puede llegar a tener la integración de dos pequeñas aplicaciones. Sin embargo, es necesario resaltar que el ejemplo de la figura es muy sencillo, que en ambientes de diseño simples se pueden requerir desde cinco o más programas de aplicación, y que la complejidad del sistema crecerá a medida que el problema que se desea resolver se vuelve más complejo.

El problema de comunicación que más se presenta al utilizar sistemas de CAD independientes en el diseño de un proyecto integral, es que en muchas ocasiones varios de los paquetes necesitan accesar los mismos datos pero los requieren en formatos y/o unidades métricas diferentes. Este problema tiene su raíz en el hecho de que, por haber sido programados como sistemas aislados, y probablemente por diferentes casas de "software", no existe homologación en la

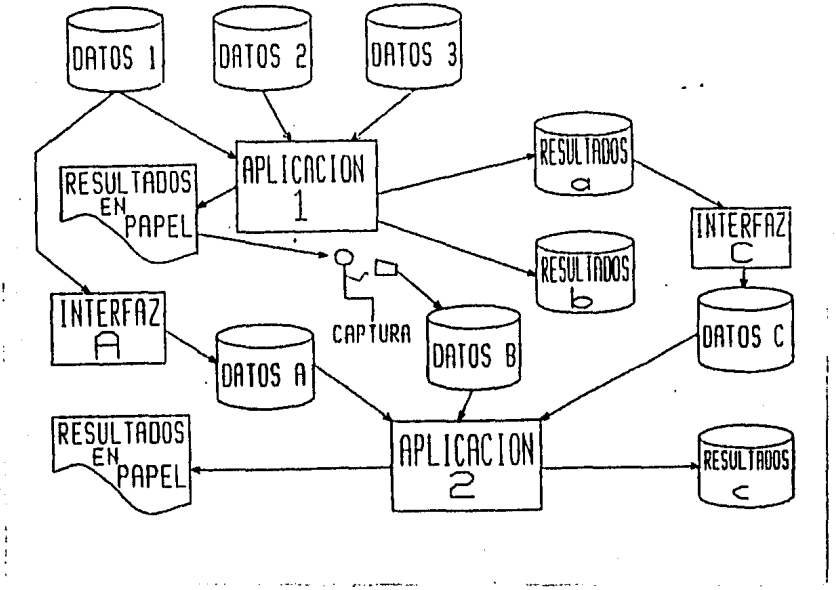


Figura 2-1. Heterogeneidad en el almacenamiento de datos

forma de representar los datos. Lo más que podemos encontrar es que cada casa de "software" use su propio esquema interno de homologación, pero sin preocuparse por lo que hayan hecho las demás casas de "software".

Los usuarios se ven obligandos resolver este problema, para lo cual tienen dos opciones posibles: a) que tengan que convertir los datos manualmente (véase en la figura 2-1 la manera en que un usuario convierte algunos datos arrojados como resultado de la aplicación 1); o b) desarrollar programas que permitan realizar las conversiones necesarias sobre los datos y/o sobre los resultados de otros programas

para que tales resultados sean utilizados por otros paquetes.

En caso que se haya desarrollado un programa para ser usado como interfaz, este únicamente sirve para convertir datos entre esas dos aplicaciones especificas, por lo que si hay otra aplicación que requiera los mismos datos hay que desarrollar otra interfaz (véase en la figura 2-1 el uso de dos interfaces diferentes causado por la falta de un sistema general que lleve el control centralizado de los datos).

Debido a que cada sistema mantiene sus propias estructuras de almacenamiento de datos incompatibles con las de los demás, cada uno de dichos sistemas almacena su propia visión del proyecto global de diseño, por lo que se propicia la repetición de datos (véase la repetición de DATOS 1 y DATOS A en la figura 2-1), con la consiguiente introducción de inconsistencias, ya sea que estas se produzcan por error humano al introducir los datos duplicados, por versiones no actualizadas de los mismos (v.gr. puede ocurrir que no se ha ejecutado la interfaz para actualizar DATOS A después de haber modificado DATOS 1), o por alguna falla de la computadora mientras alguna de las aplicaciones de CAD está realizando algún proceso.

Otras deficiencias [PROS83, LORR83], características de estos sistemas, son que generalmente adolecen de la falta de un mecanismo propio (una interfaz) para realizar consultas, captura o corrección de datos; además cuando existen, encuentramos que estos son muy primitivos si se comparan con los lenguajes de interrogación que proveen los sistemas manejadores de bases de datos. Adicionalmente, estos sistemas no cuentan con un subsistema de recuperación contra

fallas (caídas de la computadora, del programa por un evento inesperado, etcétera) ocurridas a la mitad del procesamiento y, además, la granularidad del sistema de candados (si lo hay) para asegurar la consistencia de los datos durante el acceso concurrente a los mismos se maneja a nivel de archivos completos.

También se debe tener en cuenta el problema que propicia el hecho de que cada nueva aplicación que se desarrolla posee su propia rutina para manejo de datos; por lo que un cambio en un campo o en el formato entero de representación de un tipo de dato determinado se debe reflejar realizando los cambios necesarios en cada una de las aplicaciones que accesan ese dato y/o en cada una de las rutinas que sirven como interfaces (considerando el caso que la conversión de datos entre las aplicaciones no se haga en forma manual). Durante el mantenimiento, se debe llevar estricto control de cuales son los programas que se tienen que modificar, lo cual resulta muy problemático, pero se vuelve caótico cuando son muchas las aplicaciones que requieren compartir los mismos datos.

2.2 El almacenamiento de datos de CAD usando SMBDs

El uso de las bases de datos para almacenar la información de CAD se inició como respuesta a la necesidad de que se redujeran los costos de mantenimiento y mejoramiento de los sistemas [WINP83]. Al usar bases de datos estos dos proceso se vuelven más sencillos permitiendo que los programadores dediquen la mayor parte de su esfuerzo a la mejor realización del diseño y programación de su aplicación especifica en lugar de preocuparse por la forma de almacenar y verificar sus datos y demás problemas de los que se encarga comúnmente un SMBD (referirse al capitulo 1, donde se encuentran explicaciones más detalladas).

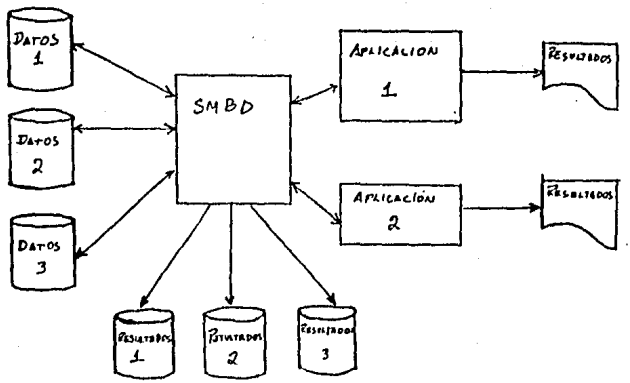


Figura 2-2. Homogeneidad en el almacenamiento de datos

En la figura 2-2 mostramos como se disminuye la complejidad estructural y funcional del ejemplo de la figura 2-1 al utilizar en su solución un SMBD, el cual se encarga de realizar en forma automática las conversiones necesarias de datos para cada aplicación. Con esto se reduce la probabilidad de inconsistencias en los datos almacenadados que pudieran ser causadas por errores humanos, omisión u olvido de ejecutar alguna de las interfaces que convierten los datos, o por copias no actualizadas de los mismos.

2.2.1 El almacenamiento usando SMBDs convencionales

Los SMBD comerciales, que han sido diseñados para soportar el manejo de datos de tipo administrativo, han sido usados de dos formas para almacenar los datos dc CAD. La primera forma intenta forzar los datos de CAD para almacenarlos usando las estructuras de datos de los SMBD existentes [GUTA82, WINP83, LORR83, UMED85]. La segunda usa los SMBD como el medio para almacenar apuntadores o referencias hacia archivos externos a la BD que son manejados por sistemas o programas externos al SMBD [UMED85].

La primera forma intenta forzar los datos de CAD para almacenarlos usando el SMBD, esto lo hacen seccionando los datos para que quepan dentro de los campos de tamaño finito que permiten manejar estos sistemas comerciales. Aunque es posible extraer datos con formato fijo de cualquier tipo de información y almacenarlos en registros, surgen varios problemas.

Primero, los SMBD convencionales no proveen el modelado semántico correcto, ni la representación del conocimiento, ni las rutinas primitivas para contestar preguntas relacionadas con los nuevos tipos de datos que debe manejar CAD. Por lo cual, el conocimiento semántico debe colocarse dentro de los programas de aplicación, lo que provoca problemas potenciales de inconsistencias y un incremento sustancial de los costos de desarrollo y mantenimiento de dichos programas de aplicación. Sin embargo, hay que señalar que el uso de las vistas ("views") de los sistemas relacionales podrían ser usados como un mecanismo, ciertamente primitivo e inadecuado, que permita manejar , en cierta forma, el significado semántico que tienen ciertos datos.

Segundo, los SMBD convencionales no proveen las estructuras de almacenamiento, ni los métodos de acceso, ni los algoritmos para el procesamiento de preguntas adecuados para ejecutar eficientemente estos programas de aplicación, lo que provoca que su ejecución sea inaceptable por lo poco eficiente [GUTA82].

Finalmente, estos sistemas no son extendibles, ya que este tipo de SMBDs no prevee los medios necesarios para permitir el acoplamiento con procesadores especiales (e.g. no permiten acoplar nuevos equipos o programas que procesen o almacenen eficientemente imágenes, o hagan modelado geométrico, etcétera) [WINP83, UMSD85].

Para que un sistema sea extendible se necesita que se le puedan agregar fácilmente nuevos sistemas procesadores de datos, es decir, debe poseer una interfaz general de acoplamiento que permita declarar la existencia de los

nuevos procesadores especiales para que las diversas partes del SMBD lo tomen en cuenta para realizar cualquier nuevo procesamiento sobre los datos.

La interfaz de acoplamiento debe poseer un lenguaje propio que permita considerar la declaración de cada nuevo procesador a los niveles de: a) almacenamiento, b) operaciones que puede realizar el nuevo procesador, y c) operaciones que tienen que tomarse en cuenta durante la optimización de preguntas.

Una vez declarado el nuevo procesador, el SMBD tomará la especifición hecha en el lenguaje de la interfaz y de ahi en adelante usará al nuevo procesador para hacer la E/S de el tipo de datos para el que fue diseñado. Y lo considerará durante las etapas de optimización local y global de preguntas para minimizar el tiempo de ejecución de la pregunta y dirigir la forma de realizar las operaciones y la recuperación del disco de los datos (véase [KORH86] para un tratamiento más extenso acerca del procesamiento y optimización de preguntas) de manera que no se degrade el tiempo de respuesta [UMED85].

En la segunda forma, para evitar algunos de estos problemas, se intenta manejar tipos múltiples de información al usar los SMBD comerciales en combinación con otras estructuras de archivos que están fuera del control de los SMBD [HASR82, LOHG83]. Tipicamente se desarrollan estructuras de archivos especiales ad hoc para los tipos de datos no convencionales; el SMBD es utilizado para almacenar los datos con tipos de formato convencional y, al mismo tiempo, los apuntadores hacia los datos que se encuentran en los archivos externos.

La información o conocimiento semántico que permite saber como conectar o asociar todos estos datos, queda como parte integral de los programas de aplicación, escritos usando el lenguaje de interrogación a la base de datos y el lenguaje anfitrión.

Este "parche" consiste en almacenar apuntadores dentro de los registros del SMBD para ayudar a localizar los datos que se encuentran en los archivos externos. Asi, las aplicaciones hacen uso del lenguaje de manipualción de la base de datos para conocer los apuntadores y los usan como parámetros para las rutinas que manejan estas estructuras en archivos externos a la base de datos.

Esta solución también presenta serios problemas: primero, el uso de archivos y datos externos totalmente fuera del control del SMBD es un regreso a la época previa a la existencia de los SMBD, incluyendo todos los problemas (que se exponen en el siguiente punto) derivados de esta falta de control; segundo, la responsabilidad de la integridad de los datos, del control de la concurrencia y de la recuperación de posibles fallas descansa sobre los programas de aplicación en vez del SMBD. La independencia entre los datos y las aplicaciones se compromete ya que al hacer cambios a la estructura de los archivos externos, para mejorar el comportamiento, se pueden invalidar algunos de los programas de aplicación existentes. Por último, la carga de la integración lógica del modelado de los datos y de la planeación global de las estrategias de acceso también queda como responsabilidad del programador de las aplicaciones.

2.2.2 El almacenamiento usando SMBDs para CAD

En la literatura se encuentran dos formas de resolver el problema de desarrollar SMBD que sean adecuados para almacenar los datos de CAD. En la primera se extienden los manejadores de bases de datos existentes agregándoles la capacidad de tratar con objetos complejos y/o con ADT, (i.e. se agregan parches al SMBD para que pueda almacenar objetos o que soporte el manejo de tipos abstractos de datos). La segunda forma se refiere a la construcción de un SMBD totalmente nuevo, pero aprovechando la experiencia que se ha tenido en el desarrollo de bases de datos y de aplicaciones de CAD con o sin el uso de los SMBD; asi como la experiencia ganada en la representación de datos y conocimientos en áreas como las de desarrollo de lenguajes, formalización de tipos de datos y sus operaciones, inteligencia artificial, etcétera. En la última solución, se han tomado al modelo orientado a objetos, al entidad-relación extendido o a alguna versión extendida del relacional como el modelo de datos a usarse en el SMBD.

Debido a que los objetos que se desean representar o describir dentro de las bases de datos son complejos y a menudo estan compuestos por otros objetos o estan asociados a otros objetos, los sistemas manejadores de bases de datos para CAD no deben limitarse a proveer el acceso a archivos, registros, relaciones o n-adas, sino que deben ser capaces de manejar el acceso a un nivel de abstracción mayor, i.e. permitir el acceso a los objetos, a sus componentes o a sus objetos asociados, encargándose del mapeo de las estructuras lógicas hacia las fisicas de cada objeto. Las operaciones que provea el sistema de almacenamiento de esta clase de SMBD deben permitir el acceso a cualquier atributo de cada

objeto, para darle los valores iniciales o modificarlos, asi como moverse a través de los diversos subojetos que componen un objeto complejo o hacia los objetos que esten asociados con el objeto que estemos analizando.

Los objetos que se encuentran formados por otros objetos requieren que las estrategias de almacenamiento aprovechen hasta donde sea posible, las relaciones estructurales de composición de los objetos, para llevar a cabo acciones especiales para almacenar los datos en disco, con el propósito de permitir el almacenamiento y recuperación eficiente de los mismos. Las acciones a tomar pueden ser: a) que se decida usar alguna técnica de agrupamiento en disco o b) que se usen técnicas especiales de indexación que permitan accesar directamente cualquier objeto.

Se han encontrado varias alternativas de solución para el manejo de objetos; unas son extensiones a algunos modelos de datos existentes y otras plantean el nuevo paradigma de objetos: el RM/T de Codd [CODE79], la representación de objetos complejos [LORR83a, STOM86], el sistema IPIP del proyecto IPAD [JOHH83] y aquellos que introducien la noción de agregados moleculares [BATD84].

2.2.2.1 El modelo RM/T. Es la extensión al modelo relacional [CODE79]. Las propiedades que permiten que este modelo pueda representar objetos complejos son:

- El uso de "surrogates"[1] (usaremos el término IDIN[2] en lugar de "surrogates" de aqui en adelante) para identificar en forma única a cada objeto en la base datos. Esto significa que el modelo ya no enfatiza la existencia de llaves primarias dentro de las n-adas.

- Permite el acceso a los metadatos en la misma forma (a través de los mismos mecanismos) que el acceso a los datos. Esto significa que establece un márco único para el acceso a datos y metadatos.

- Maneja el concepto de generalización permitiendo la clasificación de los objetos en gráficas dirigidas aciclicas de clases[3], a través de las cuales se maneja la herencia.

- Permite manejar la agregación de diferentes tipos de objetos para que formen parte de otros objetos.

Esta última propiedad (agregación) puede ser usada para indicar que un grupo de objetos debe ser tratado como un todo, lo cual se logra a través de difinir una relación de RM/T que participe en un tipo de objeto estructurado. Todas las instancias que participen en esta agregación deben ser

---

1. Los "surrogates" son identificadores internos únicos dados a los objetos, pero que son generados por el sistema.
2. IDIN significa identificador interno único.
3. Esto significa que no está restringido a jerarquias de clases como sucede con otras alternativas (véase la siguiente sección).

relacionadas a través del uso de los IDIN. El modelo, sin embargo, adolece de la falta de un mecanismo para la especificación de restricciones.

2.2.2.2 Representación de objetos complejos. El concepto de objetos complejos fue introducido por Lorie [LORR83a, STOM86] con el propósito de extender el modelo relacional para manejar los objetos que se presentan en las aplicaciones de CAD, automatización de oficinas, etcétera. Las principales propiedades de esta alternativa son:

- El uso de IDIN para distinguir entre el objeto y su representación como una n-eada dentro de la base de datos y proveer asi el medio para establecer las asociaciones entre los objetos sin utilizar el valor de la llave de la n-ada.

- Los objetos complejos estan limitados a construcciones de jerarquias en las que los objetos componentes de una jerarquia no pueden formar parte de otra.

Sin embargo esta restricción (únicamente representar jeraquías) impone limitaciones en la generalidad de uso de estos sistemas, ya que no permite manejar directamente aplicaciones de inteligencia artificial donde se comparten conocimientos, ni en la automatización de oficinas donde se requiera utilizar secciones de un mismo documento para formar otros documentos (v.gr. compartir el nombre y dirección de un cliente en diversos documentos como cartas de propaganda, facturas, pedidos, etcétera). La solución alternativa para el manejo de este tipo de aplicaciones es el viejo y problemático truco de duplicar los datos.

2.2.2.3 El sistema IPIP del proyecto IPAD. El sistema IPIP [JOHH83] fue desarrollado en la compañia BOING en relación con el proyecto IPAD y está basado en una estructura reticular del tipo CADASYL, como la definida en 1978. El sistema incluye la noción de operaciones globales[4] como la supresión de un objeto estructurado. También expande el conjunto básico de tipos proveidos por la definición de CADASYL al agregar arreglos multidimensionales ralos y densos, coordenadas para espacios de n dimensiones (dos, tres, etc), ya que se trata de tipos de datos comúnmente encontrados en las aplicaciones científicas e ingenieriles.

2.2.2.4 Agregación molecular. El concepto de agregación molecular [BATD84] fue introducido con el propósito de proveer una forma para modelar construcciones que son necesarias cuando se está trabajando con objetos complejos en una gama amplia de aplicaciones. El resultado fue un marco que distingue entre objetos ajenos y no ajenos y entre objetos recursivos y no recursivos permitiendo la aplicación de cualquiera de las cuatro combinaciones. La agregación molecular permite tomar en cuenta la necesidad de que los objetos puedan ser visibles a diferentes niveles de abstracción (debido a que este es un problema muy impotante en CAD, IA, etcétera), al establecer el marco que permite el mapeo entre diferentes niveles de abstracción y al proveer

---

4. Se definen como aquellas operaciones que al actuar sobre un objeto estructurado se propagan, es decir, propagan los efectos de la operación, hacia todos los objetos componentes del objeto sobre el que se inició la operación.

un mecanismo de operaciones globales.

2.2.3 Las operaciones en un SMBD para CAD

Otro problema que hay que resolver es el de la representación descentralizada y replicada del conocimiento semántico dentro de cada aplicación (véase la sección acerca del uso de SMBD convencionales). Esta duplicidad tiene como consecuencia que al modificar la forma de operar sobre un dato u objeto se tenga que realizar el cambio correspondiente en todas y cada una de las rutines de aplicación para asegurar que se usen los datos adecuada y homogeneamente a lo largo de todas las aplicaciones.

Por tal razón es importante evitar que el conocimiento semántico, que indica cómo tratar con cada clase de dato u objeto, se repita dentro de cada uno de los programas de aplicación que requieren utilizar los datos. Para resolver este problema se necesita que el sistema provea algún mecanismo para que las operaciones que se van a realizar sobre los datos se definan solamente una vez (como sucede cuando se define una biblioteca de programas) y que al ser usadas por todos los programas de aplicación el mencionado mecanismo se encargue de buscar la operación en la biblioteca y verificar si es una operación válida para el tipo de dato sobre el que se desea aplicar. Los mecanismos que encontramos en la literatura se engloban en dos alternativas, la primera trata del uso de "abstract data types" (ADT) [GUTA82, STOM83], la segunda se basa en el uso del modelo orientado a objetos que provee la encapsulación

de las operaciones sobre los datos como propiedad inherente del modelo [GOLA83].

Los SMBD para CAD deben proveer como operaciones mínimas las que permitan crear nuevos objetos, accesar los atributos de los objetos, y las que provean la capacidad de asociar unos objetos con otros. En las aplicaciones de CAD, de automatización de oficinas, y en algunas otras se ha encontrado la necesidad de almacenar datos con longitud variable admás de los tradicionales datos de longitud fija, por lo tanto dichas operaciones deben permitir la manipulación de datos de formato fijo y de tamaño variable. Las operaciones para asociar objetos son la respuesta que se ha dado al problema de manejar objetos que poseen un alto grado de estructuración. Por lo tanto, resulta necesario un mecanismo general que permita introducir al sistema el concepto de la estructuración de los datos, de tal manera que pueda ser usado durante la organización y recuperación de los mismos.

CAPITULO 3 EL DICCIONARIO DE DATOS PARA EL SMBD DE CAD

El propósito de este trabajo es realizar el diseño y desarrollo del sistema de almacenamiento para objetos (SALMO), que forma parte del SMBD para aplicaciones de CAD que se está desarrollando en el IIMAS [BUCA85].Sin embargo, también se hizo necesario el diseño parcial de un diccionario de datos (DD), debido a que este DD tiene incidencia directa en la definición de la información que se va a manejar en SALMO. De hecho, se tuvo que de desarrollar una interfaz que permitiera la definición de las clases de objetos que se van a manipular a través de SALMO, sin la cual no se habrían podido definir ningún tipo de dato para ser almacenado usando este sistema. El propósito de DD y de SALMO (cuya descripción se presenta en el siguiente capitulo) es permitir el almacenamiento y definición de objetos así como de asociaciones entre estos.

Este capítulo se inicia presentando información común a ambos módulos, DD y SALMO, continuá con la descripción de DD y concluye con la descripción y uso del diccionaro autocontenido de SALMO. La información común contempla la lista de parámetros de diseño, el modelo de datos y las operaciones que tienen que soportar ambos módulos. El diseño de DD se desarrolló de manera que además de soportar la definición de diversas clases de objetos, permita manejar distintas clases de asociaciones entre clases de instancias de los mismos.

3.1 Parámetros de diseño

Los parámetros que se tomaron en consideración para el diseño de estos dos módulos son los siguientes:

- Tener un ambiente orientado a objetos, lo cual permitirá modelar mejor la información de diseño. Los objetos están formados por atributos y estos pueden ser de cualquier longitud.

- Que se puedan manejar datos de cualquier longitud, fija o variable, los cuales servirán para almacenar información de cualquier tipo (numérica o alfanumérica).

- Capacidad para soportar cualquier patrón de acceso en la BD/P, con el propósito de poder modelar redes de asociaciones entre los objetos almacenados.

- Cuando se desarrollen otros módulos del SMBD para CAD, tanto el diccionario de datos como el sistema de almacenamiento deberán soportar lo siguiente:

  - En las BD/AT, un patrón de acceso (a través de las asociaciones) único para cada aplicación, que servirá para controlar el acceso rápido a la información de diseño con el objeto de proporcionar una respuesta de tipo interactiva.

  - La operación de extracción, que permita transferir datos desde la BD/P hacia la BD/AT.

- La operación de reintegración, que permita transferir las modificaciones, inserciones o supresiones hechas en la BD/AT hacia la BD/P, realizando la verificación de restricciones y excepciones.

- Control de transacciones de CAD, lo cual requiere un mecanismo centralizado que atienda cada transacción desde el momento que se inicia, por medio de una operación de extracción, hasta que termina al ejecutar exitosamente una operación de reintegración.

- Finalmente, también deben contemplarse las operaciones de inserción, supresión y modificación en la BD/AT bajo el modelo de agregación molecular [BATD84].

3.2 El modelo de datos

El modelo de datos, del SMBD para CAD, está basado en el manejo de clases e instancias de objetos, los cuales son utilizados para formar agregados moleculares [BATD84], a través del uso de instancias de diferentes clases de asociaciones. Esta característica es muy importante cuando se desea facilitar el manejo de aplicaciones de CAD, en las cuales cada artículo que se esté diseñando es modelado en forma natural si se usan asociaciones de la clase componente de. A través de estas asociaciones se puede determinar cuáles son las partes o subobjetos que constituyen cada objeto. La razón por la que se escogió dicho modelo de datos semántico fue la necesidad de tener mayor poder en el modelado de los datos [TZID82].

3.2.1 Objetos moleculares

La noción de objeto molecular puede ser entendida si usamos dos niveles de abstracción para clasificarlos; en el primero encontramos los objetos atómicos que no están formados por otros objetos; y, en el segundo, tenemos los moleculares que sí están formados por otros objetos, a su vez estos últimos pueden ser atómicos o moleculares (véase figura 3-1). Se asume que dentro del modelo se le asigna un identificador único a cada objeto al momento de ser creado, el cual es intensamente usado en el manejo de las asociaciones. Este identificador del objeto (ID_obj) está formado por la concatenación de dos atributos, el primero es el identificador de la clase a la que pertenece el objeto (ID_cla_obj) y el segundo es un número secuencial que se usa como el identificador de esa instancia del objeto.

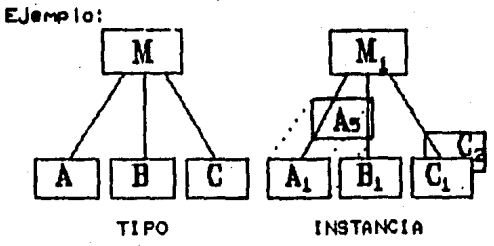


Figura 3-1. Clase e instancia de un objeto molecular

3.2.2 Clases de Objetos

La noción de clases de objetos permite que el sistema determine cuál es la estructura de cada instancia de objeto y, eventualmente, indicar cuáles son las rutinas encargadas de manipularlo. En este sentido se puede decir que se comporta a la manera de SMALLTALK, pero este comportamiento no es fortuito, ya que el SMBD para CAD se diseñó para usar un lenguaje[1] orientado al manejo de objetos, como el lenguaje de manipulación de datos de la BD, en el cual se escribirá cualquier programa de aplicación que necesite accesar la base de datos.

---

1. Un lenguaje con las características de LOOPS, C++, GALILEO o SMALLTALK serviría perfectamente para tal propósito.

Adicionalmente, el lenguaje que se utilice deberá extenderse para que contemple la definición de restricciones y de asociaciones entre los objetos, ya que los lenguajes orientados a objetos generalmente carecen de estos mecanismos y lo único que permiten es el manejo de generalizaciones [GOLA83].

3.2.3 Manejo de asociaciones

El manejo de asociaciones es inherente al modelo de datos que se emplea; así, por ejemplo, la clase asociación formado por entre cada molécula y los objetos que la componen pertenece a la clase de asociación agregación (o "aggregation"), que junto con la clase de asociación generalización (o "is-a") forman las dos clases de asociaciones que serán soportadas directamente por el SMBD para CAD.

Las asociaciones se pueden entender como una relación entre los elementos de dos conjuntos o entre dos cojuntos. En el primer caso, los elementos de cada conjunto son instancias de objetos, los conjuntos son la clase a la que pertenece cada instancia y la relación representa una asociación entre instancias. En el segundo caso la relación entre conjuntos representa una asociación de mayor nivel de abstracción que se cumple para todos los elementos de los conjuntos. Con base en esta forma de conceptualizar el modelo, se hace posible definir la inversa de dicha relación.

Smith y Smith [SMIJ77] determinaron cuales son los tipos de asociaciones o abstracciones más importantes a ser manejadas en bases de datos, pero con el fin de dar mayor claridad a este trabajo presentamos aquí las conclusiones más importantes de su trabajo y la forma como pueden ser usadas para el control del almacenamiento.

Estos autores establecieron que un SMBD debe poder representar abstracciones del tipo generalización y agregación. Generalización signigica que una clase de objeto pueda ser declarada como una superclase de otra clase, de manera que los mecanismos automáticos de herencia de atributos, operaciones y excepciones puedan ser utilizados; mientras que agregación tiene un sinúmero de acepciones, siendo las más importantes que permite definir que un objeto forma parte de otro y que un objeto está conectado con otros.

Resulta importante señalar que el modelo es capaz de representar otros requerimientos de CAD, y de otras áreas, como el de manejar un objeto a diferentes niveles de abstracción a través de la definición de asociaciones del tipo generalización, en las que cada subclase presenta la visión más especifica del siguiente nivel de abstracción.

En nuestro problema (CAD) es trascendental poder diferenciar si se trata de una agregación de objetos en la que unos forman parte de otros o de una de objetos conectados. Las razones por las que necesitamos diferenciar las clases de agregación se encuentran en: a) la definición de las políticas que rigen la agrupación de los datos en las BD/AT, y b) en el manejo de herencia de ciertos atributos, operaciones, restricciones y excepciones. La primera razón

se funda en que en las BD/AT se tendrá la necesidad de agrupar los datos de los objetos que son componentes de otros objetos, con el propósito de tener tiempos de acceso a los datos mínimos o al menos razonables. La segunda razón es la de hacer posible el manejo de diferentes heréncias a través de diferentes tipos de agregación

Por esta razón, al diseñar el sistema de almacenamiento se previó la necesidad de poder dar de alta nuevas clases de asociación cuando el usuario lo requiera. Tal es el caso de las subclases de agregación componente de o formado por que pueden tener la semántica heredada por la superclase agregación a la que pertenecen, pero además se pueden definir nuevas restricciones o excepciones especificas para esta subclase de asociación.

Es importante aclarar que dentro de nuestro modelo de datos, las asociaciones son el medio usado para mantener referencias o apuntadores a otros objetos, sólo que estos apuntadores llevan una carga semántica que permite que, bajo ellos, la existencia de los objetos tome diferentes significados (como agregación, generalización, o alguna otra más específica como conexión, parte de, etcétera). Las asociaciones son representadas dentro del sistema por medio de un identificador interno único (ID_aso). Al asociar los identificadores (ID_obj) de dos objetos con el ID_aso se logra representar la asociación entre ellos.

Por ejemplo, supóngase que se quiere asociar los componentes de un objeto usando la asociación 1, que significa 'componente de' (véase la figura 3-2).

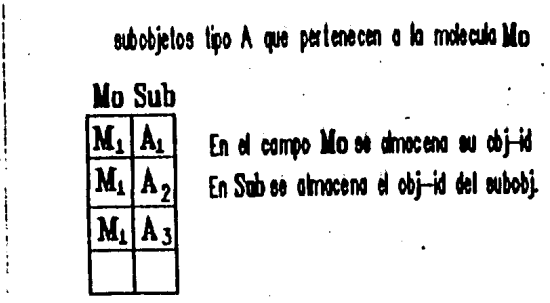


Figura 3-2. Tabla de la asociación componente de

Esta representación permite establecer varios hechos acerca de la asociación, del objeto molecular y de sus subobjetos componentes. En primer término se conoce la clase de asociación que une a los objetos en cuestión (que en este caso pertenece a la clase agregación bajo el sobrenombre componente de), pudiéndose distinguir las clases gracias al uso de una tabla distinta para llevar cada clase de asociación por cada clase de objeto. En segundo término, al almacenar los ID_obj, éstos permiten saber para cada objeto la clase a la que pertenece y la instancia en cuestión con la que se está trabajando.

Sólo nos resta indicar que las asociaciones son dirigidas (como una gráfica) por lo que si se requiere poder navegar a través de las asociaciones en ambos sentidos, se necesitará mantener tanto a la asociación y su inverso como parte del modelo del objeto que se esté representando, por lo cual cada vez que se dé de alta un nuevo tipo de asociación se tendrá que decidir si se desea definir y mantener su asociación inversa automáticamente.

3.3 Definición de clases de objetos y de asociaciones

Las operaciones básicas del diccionario de datos, permiten crear, accesar y modificar tanto las definiciones de las clases de objetos, así como de las clases de asociaciones. A estas operaciones se les puede agregar fácilmente la capacidad para declarar el manejo de restricciones y/o excepciones de integridad a nivel de toda una clase de objeto o de asociación; tal manejo es tema de otra investigación relacionada con el SMBD para CAD que esta siendo desarrollado en el IIMAS [BUCA86a], con ello el sistema podrá verificar problemas semánticos en las restricciones de diseño, librando al usuario de esa responsabilidad. Para nuestro estudio hemos separado la definición de las operaciones presentando en primer término aquéllas que sirven para crear, accesar y modificar las definiciones de las clases de objetos, en segundo aquéllas que actúan sobre las clases de asociaciones. Aunque estas operaciones se han englobado en tres tipos (inserción [o creación], modificación y consulta), algunas de ellas forman un grupo de operaciones. Así, por ejemplo, para la modificación sobre clases hay diferentes operaciones, dependiendo de si se quieren modificar las descripciones de los atributos, de las asociaciones o de las restricciones asociadas a cualquiera de los dos anteriores.

3.3.1 Operaciones para definir clases de objetos

Las operaciones para definir clases de objetos son parte de las que deben ser realizadas por el diccionario de datos. Estas operaciones sirven para describir los campos o atributos de cada clase de objeto, las clases de las posibles asociaciones con otras clases de objetos y la existencia de restricciones y excepciones para CAD. Las operaciones sobre clases que se necesitan son:

a) Crear clase de objeto (Crea_cla)

Sirve para definir para cada nueva clase de objeto su nombre, atributos, asociaciones y banderas de restricciones. Esta operación permitirá determinar, para cada atributo, su nombre y tipo, y la existencia de restricciones o excepciones para esta clase; y, para las clases de asociación, su nombre, el identificador de la asociación (ID_aso), el identificador de la clase de objeto (ID_obj) con la que se podrá asociar, y una opción que permita definir automáticamente la asociación inversa en la definición de la clase con la que se está asociando, sí así se solicita.

A la entrada se tendrá que proporcionar el nombre de la clase, la información de cada uno de los atributos y asociaciones, así como las banderas que indican la existencia de restricciones. A la salida regresa el ID de la nueva clase, el cual formará parte del identificador de cualquier instancia de la misma.

b) Modificar la definicón de una clase de objeto

Se podrá actualizar incondicionalmente la definición de una clase si no existe ninguna instancia de la misma, en caso contrario se tendrán varios tipos de actualización dependiendo de si éstas afectan o no la representación de las instancias de los objetos. En el primer caso se podrán agregar, modificar o suprimir atributos, clases de asociación y banderas de restricciones. En el segundo caso se podrán agregar nuevas clases de asociación o restricciones sin mayor problema; sin embargo, si se trata de suprimirlas, antes de hacerlo se tendrá que verificar que no existan instancias de objetos que usen estas asociaciones. En caso de querer insertar o eliminar atributos o cambiar el tipo de los mismos, primero se tendrá que crear una nueva clase de objeto con los cambios necesarios, después se copiarán los atributos de todos y cada uno de los objetos de la clase vieja hacia la nueva, y por último se borrarán la definición e instancias de la clase vieja asignando el ID_cla_obj de ésta a la nueva clase. La reutilización del identificador de la clase de objeto se hace con el propósito de seguir utilizando todas las instancias de las asociaciones que existen entre las instancias de esta clase de objeto con otras instancias. En cualquier caso se pueden modificar los nombres de los atributos y de las asociaciones. Estas aseveraciones son restricciones de consistencia de la operación por lo que deberán cumplirse siempre.

Como entrada debe proporcionársele el ID de la clase a modificar y los cambios a la misma, a la salida determina si los cambios solicitados fueron válidos.

c) Consultar la definición de una clase

Si existe la clase de objeto, usando esta operación se podrá conocer el nombre de la clase. Para cada atributo se podrá consultar su nombre, tipo y tamaño; mientras que con respecto a las asociaciones se podrá obtener el nombre, ID de la asociación e ID de la clase de objeto con la que puede asociarse. Por último, se tendrá acceso a la definición de las banderas de restricciones, tanto para atributos como para asociaciones.

La entrada estará formada por el ID de la clase, y la información de los atributos que se desean consultar. A la salida entregará un código que indicará si la operación tuvo éxito o no. En el primer caso también entregará la información solicitada.

3.3.2 Operaciones sobre clases de asociaciones

Tal como Buchmann planteó [BUCA85], el sistema deberá soportar dos clases de asociaciones en forma intrínseca, el de agregación y el de generalización. Sin embargo, para que los programadores de aplicaciones y los usuarios puedan definir nuevas asociaciones es necesario que DD y SALMO puedan realizar las siguientes operaciones que permitan el manejo y definición de diferentes tipos de asociaciones, a la vez que permitan que las aplicaciones y el sistema de almacenamiento decidan como comportarse ante las diversas clases de asociaciones.

a) Crear una clase de asociación

Permitirá dar de alta nuevos tipos de asociaciones las cuales, a partir de ese momento, podrán ser utilizados en cualquier aplicación. Antes de proceder se verificará que aun exista algún ID_aso disponible, de no ser así marcará un error.

A la entrada recibirá el nombre de la clase de asociación y los datos que indiquen si para esta asociación se tendrá que definir el inverso en forma automática y si esta asociacion es subclase de alguna otra. A la salida entregará una bandera que indicará si la operación tuvo éxito, en cuyo caso también regresará un ID_aso que identificará en forma única a ese tipo de asociación.

b) Modificar una clase de asociación

Con esta operación se podrá modificar el nombre de la asociación y la bandera que indica si se va a permitir o no la definición automática de su inverso. Antes de proceder verificará si la clase de asociación existe.

Por otro lado, en el contexto de asociaciones cambiar la clase de objeto asociada, implica dar de baja la definición anterior y, acto seguido, dar de alta la nueva asociación. Antes de proceder a realizar la operación, se tendrá que verificar que no exista ninguna instancia de la asociación a dar de baja, ya que de otra manera no se podrá realizar el cambio. Esta es una restricción de integridad y siempre se deberá respetar.

Como entrada se dará el ID_aso y el atributo a ser modificado. A la salida indicará si la operación tuvo éxito, o el motivo del fracaso.

c) Consultar una clase de asociación

Permitirá averiguar si un tipo de asociación existe, el nombre de la misma y si la inversa es automática o no.

La entrada estará formada por el ID_aso y el atributo que se desea consultar. La salida será un código que indicará si la operación tuvo éxito, en cuyo caso también entregará el valor del atributo.

3.4 Organizacion lógica del diccionario de datos

Se presenta la definición de los atributos que formarán al DD. El propósito del DD es llevar la descripción y control de cada clase de objeto y asociación que se vaya a usar en las BD/P y BD/AT a través del uso de las operaciones sobre las clases anteriormente presentadas. La definición de los campos del DD que aquí se presenta no pretende estár completa, pero incluye los campos básicos indispensables para proveer la información necesaria para llevar a cabo la definición de clases de objetos en las BD/P y AT, así como si tienen restricciones. Esto es, por el momento no se consideran los requirimientos de otras secciones de la BD de CAD (como las BDs de catálogos o plantillas), ni el manejo de resolución de homónimos y sinónimos, ya que dichos subsistemas rebasan el alcance de este trabajo.

La definición de una clase de objeto estará dada por:

```
cla_obj(ID_cla_obj,Nom_cla,Expl,#_sec_cla,Ban_res)
atr_fij(ID_cla_obj,#_atr,Nom_atr,Tipo_atr,Lng_atr,Und_atr)
atr_var(ID_cla_obj,#_atr,Nom_atr,Tipo_atr,Und_atr)
cla_aso(ID_cla_obj,ID_cla_aso,Sbr_nom_aso,±Id_aso,Auto)
op_cla(ID_cla_obj,ID_op_cla,Nom_op,Expl)
```

Y el control para establecer cuál es el siguiente ID_cla_obj a usar se lleva con:

```
id_ste_cla(Ult_id_cla)
```

Para determinar la trayectoria más usada de acceso en las BD/AT se lleva a través de la siguiente relación:

Tray(ID_obj,Id_aso,ID_Tray,Frec)

Otra información de control que debe ser llevada en el DD es la concerniente a las clases de asociaciones, para lo cual se necesitan las siguientes tablas.

meta_cla_aso(Id_cla_aso,Nom_aso,Auto,Expl,Super_cla)
id_cla_aso(Ste_id_cla_aso).

Donde el propósito de cada una de las tablas y atributos es el siguiente:

cla_obj

Es la tabla con la que se establece una nueva clase de objeto y tiene los siguientes atributos.

ID_cla_obj

Es el identificador único de esta clase, el cual deberá formar parte del ID_obj de cada una de las instancias. Además será el campo de unión entre la tabla de la clase y las tablas de atributos, asociaciones y trayectoria.

Nom_cla

Es una cadena que representa el nombre de la clase.

Expl

Es un campo, de tipo cadena de tamaño variable, que se usa para dar una explicación acerca de la nueva clase de objeto que se está definiendo.

#_sec_cla

Este atributo lleva el control del número de instancias de esta clase de objeto que han sido dadas de alta en la BD/P.

Ban_res

Este es un atributo compuesto de dos campos los cuales permiten saber si hay restricciones aplicables para esta clase de objeto ya sea porque se definieron a) sobre esta clase o alguno de sus atributos y/o asociaciones, o b) porque se hayan heredado de alguna otra clase a través de alguna asociación.

atr_fij y atr_var

Estas tablas establecen los atributos de longitud fija y variable que tiene cada una de las clases de objetos y se encuentran formadas por:

ID_cla_obj

Permite conocer a qué clase de objeto pertenece este atributo.

#_atr

Secuencia que permite establecer la posición del presente atributo dentro del objeto. Si se hace una analogía con una tabla relacional este atributo indica la columna que se está definiendo de la tabla.

Nom_atr

Cadena que representa el nombre de este atributo dentro del objeto.

Tipo_atr

Tipo de este atributo (entero, flotante, cadena, etcétera).

Lng_atr

Si el atributo es de longitud fija este campo la determina. Si el atributo es de longitud variable, entonces este campo no existe.

Und_atr

Unidades métricas en las que se encuentra el atributo.

cla_aso

La función de esta tabla es definir la posibilidad de que una clase de asociación pueda ser usada para asociar dos clases de objetos. Cada clase de asociación deberá haber sido definida previamente. Es gracias a este mecanismo que se pueden definir las clases genéricas de asociaciones (tales como agregación, generalización, etcétera) sin que cada definición quede amarrada a dos clases de objetos, pues la forma de definir que se puede usar una asociación

entre dos cla_obj es concatenando el ID de la clase de asociación con los ID de dichas clases de objetos. En síntesis se puede decir que existen a) clases y metaclases de asociaciones independientes de las clases de objetos que serán definidas a través de la tabla de clases de asociaciones, la cual será explicada más tarde; b) definiciones del uso de las clases de asociaciones que establecen la posibilidad de la asociación entre dos clases de objetos que se definen usando esta tabla y c) instancias de asociaciones que establecen la asociación entre dos instancias de objetos, y la forma en que se almacenarán será la explicada en la sección de estructuras de datos de la BD/P.

ID_cla_obj

Permite conocer de qué objeto se inicia la asociación que se está definiendo.

ID_cla_aso

Es el ID de la cla_obj al que llega la asociación establecida.

Sbr_nom_cla

Permite dar un sobrenombre a esta asociación.

±Id_aso

Es el identificador único de esta clase de asociación (Id_aso) que además es el mismo que se le dio al definir la clase de asociación. También cuenta con una bandera (±) que indica el sentido en el que va la definición de la cla_aso, la cual sirve como auxiliar para mantener las asociaciones con inverso

automático.

Auto

Permite saber si se debe declarar y mantener el inverso de esta asociación en forma automática.

op_cla

Esta tabla sirve para definir las operaciones que podrán ser usadas en las instancias de la clase de objeto que se está definiendo.

ID_cla_obj

Esta es la clase de objeto para la que se define esta operación.

ID_op_cla

Este es el identificador interno de la operación.

Nom_op

Este es el nombre de la operación.

Expl

Este atributo sirve para exponer el propósito de la operación.

id_ste_cla

Tabla con un solo atributo, mismo que sirve para llevar el control de cúal es el siguiente identificador disponible para ser usado en una nueva clase de objeto.

Ste_id_cla

Es el número secuencial que establece cuál es el siguiente ID_cla_obj que podrá ser asignado.

Tray

Cada n-eada de esta tabla sirve para construir un nodo de la trayectoria más utilizada al accesar cualquier instancia de objeto molecular, y será usada al organizar el objeto en la BD/AT.

ID_obj

Permite conocer a qué objeto pertenece este nodo de la trayectoria.

Id_aso

Determina cuál es la asociación usada para accesar el subobjeto de este objeto molecular.

ID_obj_tray

Es el ID del objeto con el que está asociado el objeto ID_obj.

Frec

En este atributo se lleva la cuenta del número de accesos que se han hecho usando esta asociación.

meta_cla_aso

Cada n-eada de esta tabla permite llevar el control de la definición de una nueva clase de asociación y consta de los siguientes atributos.

Id_cla_aso

Es el identificador único de esta clase de asociación. Se obtiene con base en el valor del atributo Ste_id_cla_aso de la tabla id_cla_aso.

Nom_aso

Nombre de esta clase de asociación.

Auto

Bandera que sirve para saber si se debe declarar y mantener el inverso de las instancias de esta asociación en forma automática.

Expl

Explicación acerca del significado de esta asociación.

Super_cla

Permite declarar si esta clase de asociación es subclase de alguna otra, i.e. define cuál es la superclase de esta clase de asociación.

id_cla_aso

Esta tabla sirve para llevar el control de cuál es el siguiente Id_cla_aso que podrá ser utilizado, permitiendo de esta manera asignar su identificador único a cada nueva clase de asociación.

Ste_id_aso

Es el atributo que lleva la cuenta de los ID de clases de asociaciones que ya han sido asignados.

Esta es la especificación de la información para control y descripción de clases de objetos y asociaciones que se debe llevar en el DD para controlar la BD/P, BD/AT y la de restricciones. En alguna etapa posterior esta definición tendrá que ser expandida para contemplar el manejo de otras BD que por ahora han sido omitidas por no haberse desarrollado todavía.

3.5 El diccionario autocontenido de SALMO

Para que SALMO funcione como un sistema general de almacenamiento se necesita que exista DD, pero debido a que DD está fuera del alcance de este trabajo y a razones de eficiencia, se decidió dotar a SALMO de un diccionario autocontenido (DA) que mantenga la información mínima necesaria para manipular diferentes clases de objetos, y de una interfaz entre DD y SALMO.

La necesidad de la existencia de DD es para que permita controlar la definición de cada una de las diversas clases de objetos y de las asociaciones. La razón de eficiencia se debe a que cada vez que SALMO haga un acceso a algun objeto necesita obtener la descripción de la clase solicitándola a DD, sin embargo, si se mantiene la sección mínima indispensable de la descripción como parte de la clase misma se ahorrará el tiempo de abrir los archivos de DD y luego realizar en él la búsqueda de la descripción.

Al dotar a SALMO con DA se logra reducir el uso de la interfaz entre DD y SALMO durante el tiempo del acceso a las clases de objetos, y sólo se requerirá su uso para la creación de cada nueva clase de objeto.

De los datos de control que DD requiere para crear una clase nueva de objeto, SALMO necesita los siguientes para poder manipular las instancias: a) el número de clase que se desea crear; b) número de atributos de tamaño fijo de esta clase; c) número de atributos de tamaño variable; d) Las posiciones en las que quedan los atributos después de reorganizar primero los de tamaño fijo y despúes los variables; e) las longitudes de los atributos de tamaño

fijo; f) el siguiente identificador interno de instancia a se usado en cada clase de objeto. Adicionalmente SALMO mantiene los tipos de los atributos de tamaño g) fijo, y h) variable previendo que tendrá que proporcionar esta información al lenguaje de manipulación de datos cuando este sea desarrollado.

3.5.1 Interfaz de SALMO y su uso

Supóngase que se va a crear la clase microcomputadora que cuenta con los siguientes atributos:

#	Nombre	Descripcion	Tipo	Longitud
1	monitor	color/B y N	fijo/char	10
2	interfaz	RGB/compuesto	variable/char	--
3	marca	IBM/compac	variable/char	--
4	modelo	PC/XT/AT	fijo/char	4
5	frecuencia	MHZ del CPU	fijo/long int	4

La interfaz de SALMO que permite la definición de la clase requiere los datos anteriormente indicados en los puntos a) al g) y que para esta clase se muestran a continuación:

a) # de la clase a crear 0.

b) # de atributos de tamaño fijo 3.

c) # de atributos de tamaño variable 2.

d) Posiciones de los atributos después de reorganizar:

posisicón al definir en DD	1	2	3	4	5
nueva posición en SALMO	1	4	5	2	3

e) Longitudes de atributos de tamaño fijo:

10	4	4

g) y h) tipos de los atributos de tamaño fijo y variable:

1	3	3	1	2

El diálogo que se realiza con el programa inip (inicia una nueva clase) que a su vez llama a la interfaz es el siguiente:

```
$inip
Da el numero de clase de objeto a iniciar: 0
Da el numero de atributos de tamano fijo: 3
Da el numero de atributos de tamano variable: 2
Da la posicion interna del atributo 1: 1
Da la posicion interna del atributo 2: 4
Da la posicion interna del atributo 3: 5
Da la posicion interna del atributo 4: 2
Da la posicion interna del atributo 5: 3
Da el tipo del atributo 1: 1
Da el tipo del atributo 2: 3
Da el tipo del atributo 3: 3
Da el tipo del atributo 4: 1
Da el tipo del atributo 5: 2
Da la longitud del atributo de tamano fijo 1: 10
Da la longitud del atributo de tamano fijo 2: 4
Da la longitud del atributo de tamano fijo 3: 4
$
```

Como resultado de ejecutar al programa inip se obtienen los archivos de datos e índices de la clase de objeto 0. Los nombres de los archivos creados son 0.d y 0.i. El contenido de los mismos se muestra a continuación usando la utilería "octal dump" (od) de UNIX.

```
Contenido inicial del archivo 0.d
0000000 000 000 000 000 000 002 000 003 000 002 000 001 000 004 000 005
0000020 000 002 000 003 000 001 000 003 000 003 000 001 000 002 000 012
0000040 000 004 000 004
0000044

Contenido explicado de 0.d
0000000 000 000 000 000 000 002¦000 003 000 002¦000 001 000 004 000 005
        6 bytes de datos de     ¦# atr   # atr  ¦ posiciones reordenadas
        control                 ¦tam fij tam var¦ de los atributos

0000020 000 002 000 003¦000 001 000 003 000 003 000 001 000 002¦000 012
 posiciones reordenadas¦Tipos de los atributos sin reordenar    ¦tamaños
 de los atributos      ¦                                        ¦

0000040 000 004 000 004
        de los atributos fijos
0000044

Contenido inicial del archivo 0.i
0000000 000 000 000 000¦ 000 000 000 001
       datos de control¦ siguiente identificador
                       ¦ de instancia a usar
0000010
```

Para terminar el ejemplo se muestra la rutina pricipal del programa inip que utiliza a la rutina de la interfaz para definir a la nueva clase de objeto.

```
main()
{
ushort
    i ,             /* contador */
    clob ,          /* clase del objeto */
    natrf ;         /* numero de atributos de tamano fijo */
    natrv ;         /* numero de atributos de tamano variable */

PG0 pg0;          /* pgn 0 del arch de datos       */

/**************************************************************
 * lee la clase de objeto asi como el numero de atributos de
 *          tamano fijo y variable que tiene
 **************************************************************/

printf("0a el numero de clase de objeto a iniciar:");
scanf(" d",&clob);
printf("Da el numero de atributos de tamano fijo:");
scanf(" d",&natrf);
pg0.afi = natrf;
printf("Da el numero de atributos de tamano variable:");
scanf(" d",&natrv);
pg0.avr = natrv;

    /* calcula el tamano para el arreglo que contendra la
     * definicion compilada de la clase de objeto a crear
     * y pidelo a malloc
     */
pg0.pps = (ushort *)malloc(sizeof(ushort)*(3*pg0.afi + 2*pg0.avr));
pg0.ptp = pg0.pps + pg0.afi +pg0.avr;
pg0.plf = pg0.pps + 2*(pg0.afi + pg0.avr);

    /*  lectura del resto de la definicion en el orden siguiente
     *       1. posicion interna de los atributos
     *       2. tipo de cada atributo
     *       3. longitud de los atrib de tamano fijo
     */
for(i=1; i <= (natrf + natrv); i++)
    {
    printf("Da la posicion interna del atributo  u:",i);
    scanf(" d",&pg0.pps[i-1]);
    }

for(i=1; i <= (natrf + natrv); i++)
    {
    printf("Da el tipo del atributo  u:",i);
    scanf(" d",&pg0.ptp[i-1]);
    }

for(i=1; i <= natrf; i++)
    {
    printf("Da la longitud del atributo de tamano fijo  u:",i);
    scanf(" d",&pg0.plf[i-1]);
    }

chk(inarch(clob,pg0));

return(NOERR);
}
```

CAPITULO 4
EL SISTEMA DE ALMACENAMIENTO

El propósito de este capítulo es describir las estructuras para almacenar datos y métodos de acceso a los mismos del sistema de almacenamiento de objetos SALMO. Este forma parte del SMBD para aplicaciones de CAD que se está desarrollando en el IIMAS [BUCA85] y permite el almacenamiento de instancias de objetos y asociaciones entre estos bajo el modelo de objetos moleculares con redes de asociaciones.

Los requerimientos de diseño, la explicación del modelo de datos y la definición de las operaciones de DD que permiten definir los tipos de datos (instancias de objetos y de asociaciones) que debe manejar SALMO fueron presentados en el capítulo anterior. Por ese motivo se hace referencia en repetidas ocasiones a lo establecido en dicho caítulo.

La presentación de SALMO está estructurada de la siguiente forma: iniciamos presentando la operaciones que SALMO debe realizar para manejar datos de los tipos definidos por DD; después, se presentan diversas alternativas que pueden ser usadas para realizar el almacenamiento de los objetos y sus asociaciones; subsecuentemente se presenta el diseño de las estructuras de datos diseñadas para soportar la alternativa de almacenamiento escogida y se termina con la explicación de los módulos principales del sistema.

4.1 Operaciones para soportar objetos

Las operaciones del sistema de almacenamiento SALMO, permiten crear, accesar y modificar instancias tanto de las clases de objetos como de las clases de asociaciones. A estas operaciones se les puede agregar fácilmente el manejo de restricciones y/o excepciones de integridad a nivel de instancias particulares que es tema de otra investigación relacionada con el SMBD para CAD que esta siendo desarrollado en el IIMAS [BUCA86a], con ello el sistema podrá verificar problemas semánticos en las restricciones de diseño, auxiliando al usuario en esa responsabilidad. Para nuestro estudio hemos separado la definición de las operaciones presentando en primer término aquéllas que sirven para insertar, accesar y modificar objetos y en segundo las que operan sobre las asociaciones[1],

4.1.1 Operaciones sobre objetos

Estas operaciones realizadas por SALMO permiten insertar, consultar y modificar instancias de objetos. Se podrá escoger si se desea hacer el acceso a cierto campo en especial o a todos los campos del objeto, lo cual incluye las banderas de restricciones/excepciones. Tal como ya se ha aclarado con anterioridad, estas son operaciones primitivas que únicamente saben almacenar, recuperar y actualizar

---

1. Cuando se hable acerca de objetos o asociaciones, se está hablando de sus instancias, no de sus clases.

información de una instancia. Es por esto que deberán existir otras operaciones similares pero de mayor nivel que permitan hacer uso de la verificación de restricciones o excepciones de CAD, o que manejen objetos en su forma molecular, lo cuál significa que deberán saber controlar el acceso a todos y cada uno de los subobjetos que formen parte de la molécula. Como estas últimas operaciones llevarán ésta y otras cargas semánticas han sido definidas para ser realizadas en otras etapas (dichas acciones son parte de otras tesis de maestría), lo cual permite dedicar todo el esfuerzo presente al desarrollo de la etapa de almacenamiento puro.

a) Insertar un instancia de una clase de objeto

Mediante esta operación se puede crear una nueva instancia de una clase de objetos, al mismo tiempo que se proporcionan valores a los atributos de la misma. Lo primero que se hará será verificar la existencia de la clase, si existe se procederá a dar de alta la instancia con los valores de los atributos dados, de otra forma se produce un error.

Como entrada se proporcionarán el ID de la clase, la información de las restricciones para esta instancia particular y los valores de los atributos. A la salida se obtendrá el ID de la instancia o un código de error.

b) Modificar un instancia de una clase de objeto

Servirá para modificar uno o varios de los atributos de un objeto, incluyendo las banderas de las restricciones. Antes de proceder a realizar cualquier cambio se verificará la existencia de la instancia indicando un error en caso de no encontrarla.

A la entrada debe recibir el ID de la instancia, los atributos y banderas de restricciones a ser cambiados y sus nuevos valores. Como salida entregará el ID de la instancia o un código de error en caso de no poder realizar las modificaciones requeridas.

c) Consultar un instancia de una clase de objeto

Realmente esta no es una sola operación, sino que es un grupo de operaciones, la primera de las cuales permitirá obtener el valor de cada uno de los atributos de una instancia, la segunda permitirá conocer la longitud de un atributo cuando éste sea de longitud variable. Al igual que en la anterior operación, lo primero será verificar la existencia de la instancia.

A la entrada se deberá proporcionar el ID de la instancia y el atributo a consultar. A la salida regresará el valor o longitud del atributo consultado o un código de error en caso de presentarse algún problema.

4.1.2 Operaciones sobre asociaciones

Estas operaciones servirán para insertar, consultar y modificar instancias de asociaciones, y únicamente tienen sentido cuando se aplican sobre instancias de objetos, es decir, que una instancia de una asociación sólo puede existir cuando está asociando a dos instancias de alguna clase de objeto. Por otro lado, se da por hecho que en la definición de la clase del objeto exista la asociación que se quiere utilizar. De no ser asi, no se podrán definir instancias de asociaciones hasta no dar de alta dicha definición.

a) Insertar una instancia de asociación

Insertar una instancia de asociación significa asociar dos instancias de objetos. Por supuesto primero se verficará si está determinada la posibilidad de asociación entre ambas clases a través de una consulta a la definición de la clase del objeto. Si existe la definición de la asociación y además en ella se establece que debe crearse automáticamete el inverso de la misma, se darán de alta la asociación y su inverso, de otra manera sólo se dará de alta la asociación.

A la entrada se proporcionarán los IDs de las dos instancias y el tipo de asociación. A la salida un código que indica si se pudo realizar o no la operación.

b) Modificar una instancia de asociación

Esta operación no tiene sentido en el contexto de asociaciones, puesto que modificar una asociación significa dar de baja una asociación ya existente con una clase C1 para luego proceder a dar de alta una nueva asociación pero con una clase diferente C2. Este procedimiento claramente deja ver que lo que se tiene que hacer es dar de baja la primera asociación y luego insertar la segunda permitiendo así que la operación de inserción sea la que se encargue de toda la verificación que implica el alta de una nueva asociación.

c) Consultar una instancia de asociación

Esta operación permitirá conocer si existe una asociación entre dos instancias de objetos. Si existen varias asociaciones se entregará la siguiente por cada invocación.

Como entrada se le proporcionará el ID_aso y el ID de la primera instancia de objeto. A la salida entregará un código que indicará si hubo éxito, en cuyo caso también entregará el ID de la segunda instancia de objeto.

4.2 Alternativas para la organización del almacenamiento

Teniendo en consideración los parámetros expuestos en el capítulo anterior, se presenta el estudio realizado sobre algunas alternativas de almacenamiento que pueden ser usadas en CAD para el manejo de objetos complejos y sus asociaciones. Estas alternativas permiten soportar en mayor o menor medida la carga semántica de los datos a través del uso de asociaciones entre los objetos.

El estudio incluye las ventajas y desventajas de cada alternativa. Después se presenta la discusión que muestra las razones que sirvieron para descartar algunas y escoger otras al aplicarlas en los contextos de las BD/AT y BD/P.

En la figura 4-1.a se presenta la definición de los atributos del objeto molecular ‹M› y de sus subobjetos atómicos tipo ‹A›, ‹B› y ‹C› (recordar figura 3-1), así como los atributos que forman a cada uno de dichos objetos y subobjetos. Este ejemplo será utilizado a través de esta sección para mostrar el uso de cada una de las alternativas aquí explicadas.

4.2.1 Tablas planas y relaciones binarias

Las instancias de los objetos se almacenarán en tablas planas normalizadas. Se tendrá una tabla por cada clase de objeto (figura 4-2), los atributos de cada tabla serán definidos en forma individual para cada clase de objeto (figura 4-1.a). Cada interrelación o asociación se

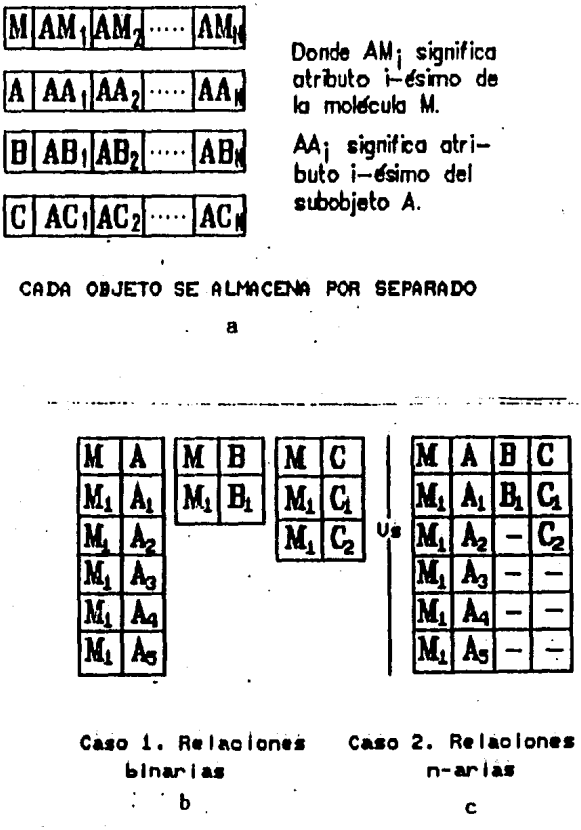


Figura 4-1. Definición y representación del objeto M

representará por medio de una tabla binaria de los identificadores de objetos (ID_obj) (figura 4-1.b) que fungirá como índice de unión binario. Cuando se necesite saber cuáles son los objetos componentes de otro, se buscará en cada una de las tablas binarias las asociaciones correspondientes.

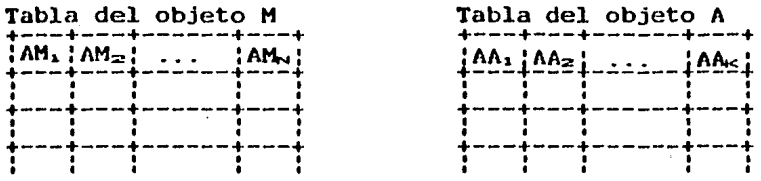


Figura 4-2. Tabla de cualquier objeto molecular o atómico

La principal ventaja de este mecanismo es su flexibilidad, ya que permite definir nuevos objetos moleculares en forma dinámica y que las asociaciones entre los objetos sean redes i.e. no se limita a manejar árboles de asociaciones. Además, para los objetos consistentes de un número variable de diferentes subobjetos, este mecanismo resulta económico en espacio de almacenamiento, dado que elimina los valores nulos. La principal desventaja se debe a los múltiples accesos a las tablas binarias para extraer todo un objeto, ya cada tabla sólo guarda las referencias a los subobjetos de una clase particular, por lo que se tiene una tabla por cada pareja de objetos que se necesiten asociar (v.gr. una tabla para las asociaciones entre instancias de las clases M y A, otra entre M y B, etc), a su vez para cada una de las clases de dichos subobjetos se deberá tener otra tabla binaria que modele la asociación de agregación hacia sus propios subojetos.

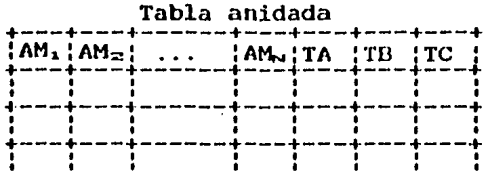


Figura 4-3. Tabla anidada del objeto M y sus objetos componentes

4.2.2 Tablas planas con relaciones n-arias

Este mecanismo de almacenamiento es similar al anterior, pero pretende corregir el problema de accesos múltiples a las diversas tablas de asociaciones binarias a través de la definición de una sóla tabla n-aria (figura 4-1.c), por cada asociación, que contenga los identificadores de los objetos que componen al objeto molecular. La desventaja principal de este mecanismo es su rigidez, debido a que no resulta fácil extender la definición de las asociaciones desde un objeto molecular hacia nuevas clases de objetos, ya que se tendria que agregar una nueva columna a la tabla por cada nueva clase de subobjeto. Además, si dentro de un objeto molecular existe un número variable de subobjetos de cada una de las clases, provoca el almacenamiento de demasiados espacios nulos en la relación n-aria (figura 4-1.c), ya que se debe reservar el espacio para almacenar el máximo número de posibles clases de subobjetos y varias de las columnas quedarian vacias. Aunque esta solución reduce el número de tablas binarias al permitir modelar la asociación entre una clase con varias clases, de todas maneras se siguen requiriendo otras tablas

distintas para modelar los subobjetos de los subobjetos.

4.2.3 Tablas anidadas

Las tablas anidadas son aquellas tablas en las que algunos de sus atributos son a su vez otras tablas completas (figura 4-3), y se prestan para el almacenamiento de objetos jerárquicos ya que permiten que los subobjetos atómicos se almacenen como parte de cada objeto molecular del que formen parte.

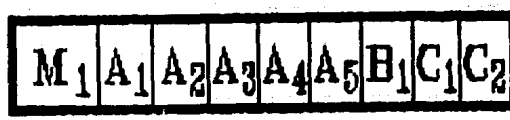


Figura 4-4. Almacenamiento secuencial en preorden de la jerarquía

La jerarquía modelada se puede almacenar secuencialmente en preorden (figura 4-4) o usando listas ligadas en preorden (figura 4-5). La ventaja de esta alternativa estriba en que se presta para que la información de la jerarquía quede agrupada en el disco, favoreciendo que la recuperación de los datos sea muy rápida. Por el contrario, una desventaja es la limitación para representar sólo objetos con estructura jerárquica, otra desventaja, que se presenta cuando se almacena el objeto secuencialmente en preoden, es que si se necesita modificar algún atributo o

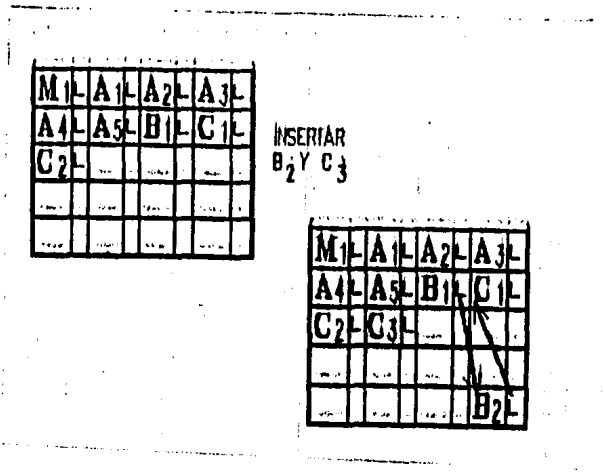


Figura 4-5. Almacenamiento con ligas de la jerarquía

insertar o eliminar subobjetos, hay que deslizar todos los datos desde el punto donde se realiza la operación. Este problema se puede atenuar un poco agregando a cada uno de los atributos y objetos un campo de apuntador para que el acceso al siguiente atributo se haga siguiendo la liga.

4.2.4 Tablas anidadas con índice de unión

Esta alternativa es un híbrido de los anteriores, se caracteriza por que se almacenan los objetos separados unos de otros (uno por cada tabla), pero se mantenienе un índice de unión con identificadores de los objetos (o sea mantiene asociaciones en forma de árbol), el cual permite localizar a cada uno de los subobjetos de un objeto. Esta alternativa

tiene la ventaja de conservar normalizadas las tablas de los objetos atómicos, a la vez que evita la búsqueda innecesaria de las asociaciones en múltiples relaciones binarias; también facilita la realización de inserciones y supresiones. Las desventajas se deben a que el manejo de tablas anidadas restringe la representación a jerarquías de objetos y a que los objetos se almacenan separados por lo que hay que realizar múltiples accesos.

4.3 Las estructuras de datos para la BD/P y BD/AT

Tal como se explicó en el capítulo anterior, los datos de cada diseño se representan utilizando objetos complejos y asociaciones entre ellos. A continuación se presenta la organización física, para la alternativa de almacenamiento escogida, diseñada para representar la información de los objetos y sus asociaciones. Las estructuras están organizadas de tal forma que permiten distinguir, durante el acceso, clases de objetos, instancias particulares de cada clase de objeto, clases de asociaciones e instancias particulares de cualquier clase de asociación.

La alternativa de almacenamiento escogida es la que usa tablas binarias para las asociaciones y tablas planas para los objetos. Esta alternativa permite representar fácilmente los objetos y redes de asociaciones entre estos. Los esquemas de tablas anidadas resultaron eliminados por no poseer esta capacidad y limitarse a represntar sólo jerarquías.

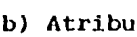
A nuestro diseño le hicimos algunas mejoras que aceleran el acceso a las asociaciones que se encuentran en las tablas binarias. Todas razones por las que se escogió esta alternativa y los motivos que orillaron a que se le hicieran mejoras, se presentan en el siguiente capítulo en la sección de discusión de las alternativas escogidas.

4.3.1 Las estructuras para los objetos

Los objetos están formados por atributos, los cuales sirven para almacenar información de cualquier tipo, desde numérica hasta textos. Se hace notar que en este sistema los atributos pueden ser de longitud fija o variable lo que lo hace totalmente diferente de los sistemas convencionales en los que las longitudes siempre son fijas y por lo tanto previamente determinadas (figuras 4-6.a y 4-6.b).

Por otro lado, se decidió proveer a cada instancia con un identificador interno único (Id_obj). El propósito de este identificador es utilizarlo como la llave primaria para realizar cualquier acceso a través de las operaciones sobre objetos y asociaciones (referirse al capítulo anterior). Adicionalmente, este identificador interno permite eliminar los problemas provocados por el uso de llaves definidas y controladas por el usuario.

a) Atributo de tamaño fijo con seis posiciones



b) Atributo de tamaño variable

Figura 4-6. Formato de los atributos

Para diseñar la forma del Id_obj se buscó que tuviera las siguientes características:

a. Que sea un identificador estable, es decir, que no se vea afectado cuando se tengan que hacer reorganizaciones físicas de la BD.

b. Que permita copiar objetos complejos (v.gr. entre la BD/P y las BD/AT y viceversa) sin tener que modificar las referencias hacia el objeto en cada donde se usen. Esto significa que lo único que es diferente entre una instancia de objeto y su copia son sus direcciones físicas.

c. Que permita realizar el acceso directo a los atributos de los objetos sin tener que seguir ninguna lista de apuntadores o de superobjetos que apuntan a subobjetos. Esto tiene el propósito de evitar el problema de tener que leer todos los objetos que están antes que el que se intenta accesar cuando dicho objeto forma parte de una jerarquía.

Al considerar lo anterior se decidió que el identificador de cada instancia de objeto fuera algún número secuencial y que se tuviera alguna forma de mapear de ese número hacia la posición física de la instancia dentro de la BD. El resultado del diseño se expone a continuación.

El identificador interno de cada objeto está compuesto por la concatenación de dos atributos, el primero es un número que indica la clase de objeto a la que pertenece la instancia de objeto con la que se está trabajando, el segundo es otro número que identifica en forma única a esa

instancia particular. Como se aprecia, este identificador permite conocer para cada instancia su clase y su número particular de instancia dentro de esa clase (véase figura 4-7).

[Id_cla_obj ¦ #_sequencial_de_instancia]

Figura 4-7. Composición del identificador único Id_obj

Para almacenar los objetos se decidió utilizar dos archivos por cada clase, en el primero se agrupan los datos de todas las instancias de una clase de objeto particular, en el segundo se mantiene una tabla de acceso que contiene un apuntador por cada instancia creada y cuyo índice de entrada es el número secuencial de la instancia mostrado en la figura 4-9. Este apuntador determina la posición inicial de cada instancia que se encuentra en el primer archivo (véase figura 4-8). Una vez que una instancia ha sobrepasado el tamaño de una página de disco, podrá seguir creciendo conforme se necesite, solicitando más páginas de 512 bytes.

De esta manera se hace posible manejar al Id_obj como un identificador independiente de la posición física que ocupa la instancia en el disco.

La desventaja de esta solución es la necesidad de espacio adicional para mantener la tabla de acceso que

permite realizar el mapeo identificador -› dirección que en notación funcional se define como:

dir_obj(Id_obj) : Id_obj -› dirección física

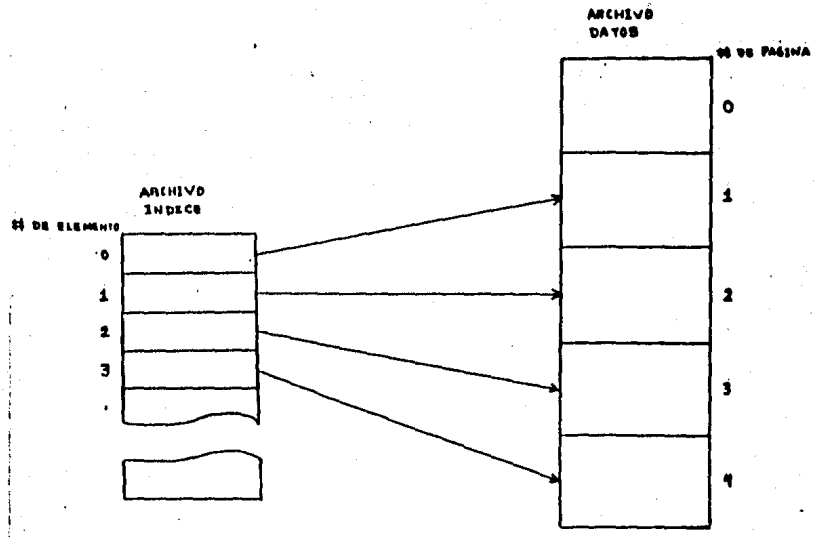


Figura 4-8. El almacenamiento para cada clase de objeto

4.3.1.1 El archivo de datos La estructura interna del primer tipo de archivo, al que llamaremos archivo de datos de aquí en adelante, está compuesta por los datos de control y los valores de los atributos. Los datos de control están organizados de la siguiente manera, existe un área de control general que es válida para todas las instancias de la clase y un área de control particular para cada instancia.

El área de control general se inicia en la página cero de cada archivo de datos, la información que contiene este bloque debe estar bajo el control del diccionario de datos, pero debido a que dicho módulo aún no ha sido desarrollado, se tuvo que almacenar y controlar aquí para permitir la manipulación de clases de objetos. Los campos que forman esta página son mostrados en la figura 4-9 y se explican a continuación.

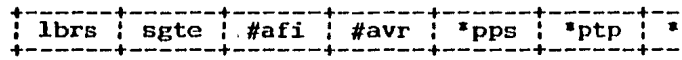


Figura 4-9. Campos de la página cero de cada la clase

a. El apuntador a la lista de páginas libres, es decir aquellas que han sido borradas, su valor inicial apunta a VACIO. Esta dirección se representa usando tres bytes. La razón de que sólo se necesitan tres bytes es por que UNIX únicamente permite direccionar 2^{32} bytes por archivo. Puesto que lo que se desea es

apuntar hacia páginas de disco de 512 = 2^9 bytes, para saber cuantos bits necesito para almacenar estos apuntadores se hace la división $2^{32}/2^9$ que da por resultado 2^{23}, valor que cabe perfectamente en tres bytes.

b. Otro apuntador hacia el siguiente bloque de disco que estando al final del archivo nunca ha sido usado. Su valor inicial apunta al bloque 1, y se almacena usando tres bytes como en el caso anterior.

c. El número de atributos de tamaño fijo. Se decidió que este se almacenara en dos bytes permitiendo así que cada objeto pudiera tener 2^{16} diferentes atributos de tamaño fijo, sin contar los atributos de tamaño variable. Usando este valor y el del número de atributos de tamaño variable se calcula el número de páginas del archivo que usa esta área de control.

d. El número de atributos de tamaño variable. Se decidió que este se almacenara en dos bytes permitiendo así que cada objeto pudiera tener 2^{16} diferentes atributos de tamaño variable.

e. El arreglo que indica la posición interna en la que quedó cada atributo. Este arreglo se hizo necesario por razones de eficiencia, ya que se decidió agrupar por un lado los atributos de tamaño fijo y por otro los de longitud variable, con lo que se disminuye la cantidad de información que hay que mover cuando ocurran modificaciones a los atributos de tamaño variable.

f. El arreglo con los tipos de cada atributo. Se incluyen tanto los tipos de los atributos de longitud fija como los de variable. Se utilizan dos bytes por cada atributo, lo que significa que es posible que se definan hasta 2^{16} tipos de atributos diferentes.

g. El arreglo con las longitudes de los atributos de longitud fija. Se usan dos bytes por atributo, por lo que el tamaño máximo de cualquier atributo de longitud fija es $2^{16}-1$ bytes.

El área de control particular, de cada instancia de objeto, contiene la información indispensable para accesar los atributos de cada objeto. Esta información funciona como un directorio particular de acceso, en el que las direcciones de los atributos son almacenadas en forma relativa a las direcciones absolutas de las páginas que ocupa dentro del archivo. Para convertir la dirección relativa de un atributo a su dirección absoluta dentro del archivo de datos se mantiene también una tabla directorio en la que se encuentran las direcciones de las páginas del archivo que son ocupadas por la instancia con la que se está trabajando. La información almacenada aquí depende del tipo de instancia , ya que se definieron dos tipos básicos para hacer mejor uso del espacio en disco. El primer tipo, el más general, lo forman aquellos objetos de tamaño mayor a una página de disco, el segundo tipo está constituido por aquellos objetos menores a una página. Los campos de esta área de control (que pueden existir o no dependiendo del tamaño del objeto) son explicados a continuación.

a. El número de páginas que usa esta instancia sin contar esta primera página que demominamos la página cero de la instancia. Se almacena en tres bytes por la razón previamente explicada.

b. El apuntador al primer byte de datos. Ocupa cuatro bytes.

c. La tabla directorio que contiene los siguientes datos cuando su tamaño es diferente de cero:

   1. El número de páginas de esta instancia que son usadas para almacenar la tabla directorio. Se usan tres bytes.

   2. Un apuntador por cada página que es usada por esta instancia a excepción de la primera. Cada uno ocupa tres bytes.

d. El arreglo de apuntadores a los atributos de tamaño fijo y longitud variable. Cada apuntador se almacenena usando cuatro bytes, lo que permite que cualquier objeto particular pueda ser del mismo tamaño máximo que UNIX permite para los archivos (2^{32} bytes).

e. El arreglo que contiene las longitudes de los atributos de longitud variable. Cada elemento ocupa cuatro bytes por la razón expuesta en el punto anterior.

f. Los valores de los atributos de tamaño fijo se almacenan a continuación. Cada uno usa la cantidad de bytes especificada en el área de contol general.

g. Los atributos de tamaño variable. Cada uno usa la cantidad de bytes especificada en el arreglo de las longitudes de los atributos de tamaño variable de esta instancia.

4.3.1.2 El archivo de índices La organización del segundo archivo (o archivo de índices de aquí en adelante) es mucho más sencilla. Cuenta con dos áreas, la primera es de control, ocupa los primeros ocho bytes del archivo, de los cuales los cuatro primeros contienen un apuntador hacia las áreas borradas, su valor inical es VACIO y los cuatro últimos están apuntan a la siguiente área que estando al final del archivo nunca ha sido usada, su valor inicial es 1L (figuras 4-10.a y 4-10.b respectivamente).

Figura 4-10. El área de control del archivo de índices

El resto del archivo contiene un arreglo de registos donde el índice al arreglo es el número particular de instancia que forma parte del identificador único de objeto Id_obj. Cada uno de los registros de este arreglo consta de cuatro campos (figura 4-11) , el primero, de cuatro bytes, es al apuntador hacia la página del archivo donde se encuentra la información inicial del objeto; el segundo, de dos bytes es un contador que servirá para que el diccionario de datos lleve la información de la trayectoria más usada para accesar un objeto y sus componentes; el tercero es un byte que se ha reservado para su uso futuro por el sistema

de control de restricciones y excepciones; y el cuarto determina el tipo de objeto de bajo nivel, por medio del cual se puede llamar al procesador adecuado de bajo nivel para manejar eficientemente la E/S de ese objeto.

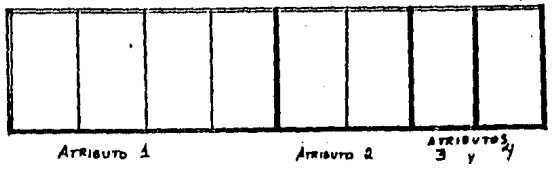


Figura 4-11. El registro para accesos del archivo de índices

4.3.2 Las estructuras para las asociaciones

Las diferentes clases de asociaciones que existen entre los objetos de diseño y entre sus partes se representan formando asociaciones binarias (parejas odenadas) con los identificadores internos de los objetos. Será la responsabilidad del diccionario de datos (cuando se desarrolle) el permitir crear diversas clases de asociaciones entre los objetos almacenados definiendo asociaciones como parte_de, es_un o conectado_a. Las asociaciones binarias se almacenan usando un árbol B con bloques o páginas de disco de 512 bytes. Se tiene una página de control o página cabeza del árbol, la cual siempre se encuentra en el bloque cero del archivo de cada asociación. Y las demás son las páginas normales del árbol B [BAYR72, WIRN76].

La página de control contiene la siguiente información, la dirección de la página raiz del árbol, en cuatro bytes; el número máximo de llaves que caben en una página, en dos bytes; un apuntador a la lista de páginas desocupadas, en cuatro bytes; un apuntador al final del archivo que indica la dirección de la siguiente página que puede ser usada si no hay ninguna en la lista de borradas, en cuatro bytes; una bandera que permite manejar candados para indicar que el árbol ya está en uso por otra aplicación, en dos bytes; un apuntador a la página más a la izquierda del árbol y otro a la página más a la derecha del mismo, cada uno ocupa cuatro bytes.

La organización de cada página del árbol B es la siguiente: una bandera que ocupa dos bytes para determinar si la página está en las hojas o en las ramas del árbol, un apuntador hacia la página padre que ocupa cuatro bytes, un apuntador hacia la página hermana a la izquierda que ocupa cuatro bytes, un apuntador hacia la página hermana a la derecha que ocupa cuatro bytes, el número de llaves que se han almacenado en la página que ocupa dos bytes, el apuntador hacia la página hija cero que ocupa cuatro bytes y el arreglo que contiene parejas de llaves y apuntador hacia otras páginas hijas cada uno de ellos ocupa dieciseis bytes, doce son de la llave y cuatro de cada apuntador.

4.3.3 Las estructuras para el manejo de archivos y buffers

Debido a que UNIX limita el número de archivos que se pueden mantener abiertos y que en cada aplicación de CAD se usan muchos objetos de diferentes clases, se vió que era necesario contar con una forma automática de cerrar archivos que tuvieran poco uso. Para llevar el control de archivos se definió una estructura que consta de los siguientes campos.

inda Bandera que indica la clase de archivo de que se trata (índices, datos, asociaciones).

nom Cadena que contiene el nombre del archivo.

modo Modo en el que se encuentra abierto el archivo.

bloq El número del bloque que está en el buffer.

sc El código que fue regresado por la más reciente lectura o escritura, el cual actua indicando el número de bytes usados dentro del buffer.

buf El buffer para el bloque indicado en "bloq".

ban Una bandera que indica si el buffer se encuentra limpio o sucio, es decir, si ya se ha realizado alguna escritura sobre el buffer.

tiempo El sello de tiempo para determinar el archivo a cerrar usando el método del menos recientemente utilizado.

Ademas se diseñó un esquema de manejo de buffers a diferentes niveles, de archivos, de objetos y árboles B. El primer nivel ya se mostró en el campo buf del control de E/S de archivos. El manejo de "buffers" a nivel de objetos está diseñado para que permita mantener en memoria la descripción general de la página cero de un número (limitado por el tamaño de una tabla de dispersiones) amplio de clases de objetos y de la página cero de la última instancia sobre la que se realizó algún proceso de E/S en cada clase de objeto.

Mantener en los "buffers" los descriptores generales de diversas clases de objetos implica que esta información solo se necesita leer del disco en caso que no esté ya en la tabla de "buffers".

Para los árboles B, también se mantienen en memoria los descriptores de cada archivo de asociaciones que se encuentre abierto asi como la página del árbol sobre la que se está haciendo la E/S.

4.4 Los módulos del sistema

El sistema se ha separado en a grandes razgos en las rutinas de alto nivel que operan sobre los objetos y sobre las asociaciones, y las de bajo nivel que se encargan de abrir o cerrar los archivo y del manejo de los "buffers". Las caracteristicas sobresalientes de las rutinas se describen a continuación, incluyendo la explicación de algunas decisiones de diseño tomadas durante el desarrollo.

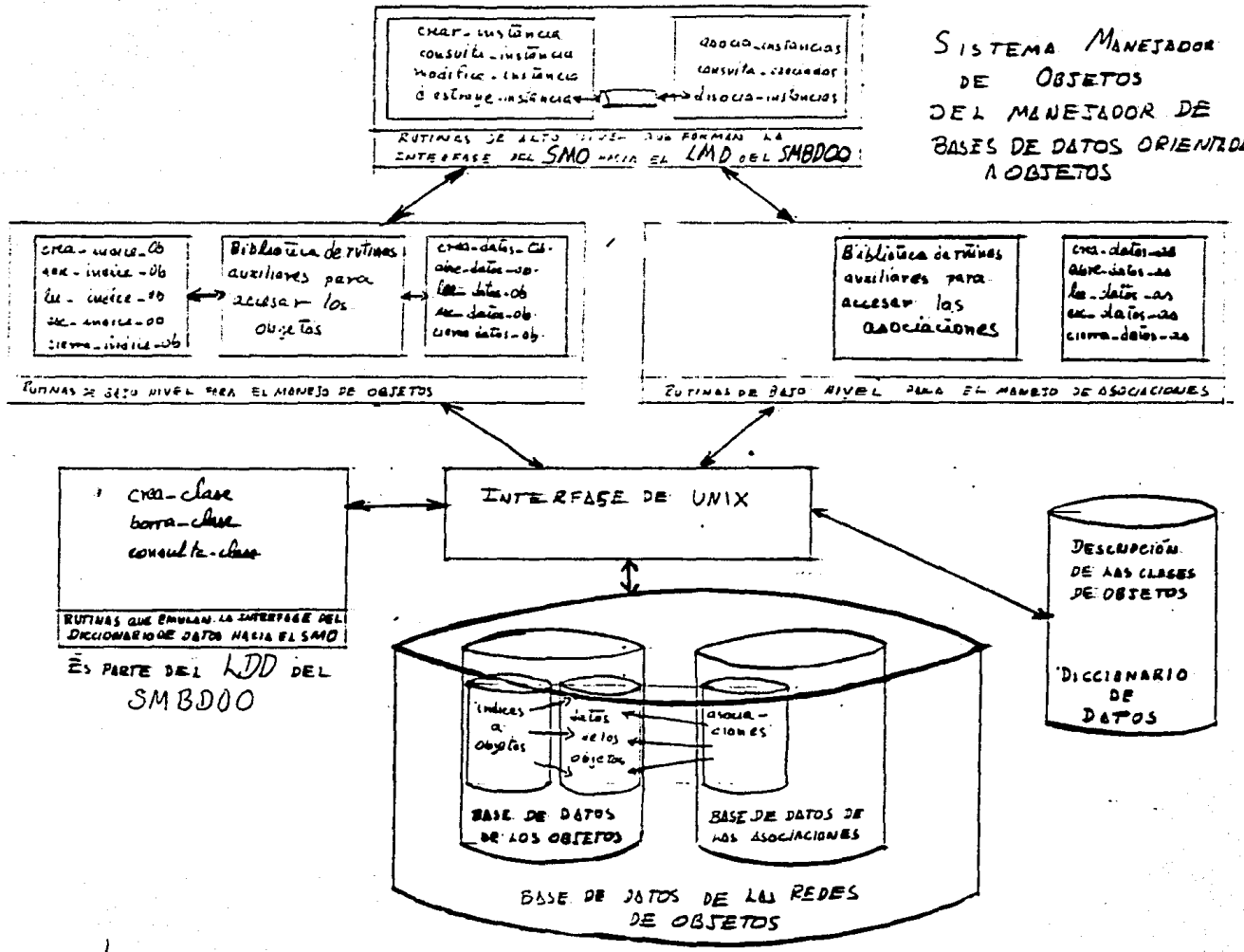
4.4.1 La rutinas, de alto nivel, para accesar los objetos

Son aquellas que permiten trabajar sobre las instancias de los objetos para crearlas; modificar o accesar cualquiera de sus atributos y por último, solicitar información, como la longitud, acerca de alguno de los atributos. A continuación se explica lo que hacen las dos rutinas más importantes.

4.4.1.1 La rutina que crea instancias de objetos realiza los siguientes pasos:

I. Obtiene el identificador para la nueva instancia, para lo cual hace uso del área de control del índice para la clase de la instancia en cuestión.

II. Carga en memoria la descripción general de la clase que se encuentra en la página cero del archivo de datos (verifica primero que no se encuentre en la tabla de buffers de descriptores de objetos).



SISTEMA MANEJADOR DE OBJETOS DEL MANEJADOR DE BASES DE DATOS ORIENTADOS A OBJETOS

III. Obtiene el primer bloque libre del archivo de datos donde se almacenará la instancia.

IV. Carga en la entrada correspondiente del índice la dirección de ese bloque.

V. Construye el esqueleto inicial de la instancia, lo cual incluye la página de control con las longitudes de los atributos de tamaño variable iguales a cero y los valores iniciales de los apuntadores y tablas; también se almacenan los espacios para los atributos de longitud fija con valor inicial nulo.

VI. Se escribe el esqueleto al disco iniciando la operación en el bloque al que se hace referencia dos puntos atrás.

VII. Finalmente, se actualizan los apuntadores de bloques libres en la página cero del archivo de datos.

El resultado de aplicar esta rutina para crear una instancia vacia para la clase de objeto 0 que hemos usado como ejemplo (véase la sección 3.5.1) se puede apreciar en el vaciado octal de los archivos 0.d y 0.i.

Contenido de 0.d después de crear una instancia

```
0000000 000 000 000 000 000 004 000 003 000 002 000 001 000 004 000 005
0000020 000 002 000 003 000 001 000 003 000 003 000 001 000 002 000 012
0000040 000 004 000 004 000 000 000 000 000 000 000 000 000 000 000 000
0000060 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
*
0001000 000 000 000 000 000 000 002 044 000 000 000 000 000 000 000 012
0001020 000 000 000 016 000 000 000 000 000 000 000 000 000 000 000 000
0001040 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
*
0001066
```

En el vaciado del archivo 0.d se pueden apreciar los siguiente cambios respecto al archivo 0.d de la sección 3.5.1:

1. En los datos de control del byte 3 al 5, el apuntador al siguiente bloque disponible apunta al bloque con dirección 1024_{10}. (sólo se almacenan los tres bytes más significativo de la dirección que tiene cuatro bytes)

2. En el bloque con dirección 512_{10} ó 01000_8 muestra los siguientes datos de control de la instancia y los atributos con valores nulos:
   a) cero paginas adicionales de esta instancia en cuato bytes.
   b) el primer byte de datos se encuaentra en la dirección 002_8, $044_8 = 02_{16}$, $24_{16} = 1044_8 = 548_{10}$. Ocupa cuatro bytes.
   c) no hay tabla directorio porque el campo a) es cero.
   d) el arreglo de apuntadores a lo atributos de longitud fija y variable queda de la siguiente manera:
   Las direcciones de los tres atributos de long fija 000 000 000 000, 000 000 000 012, 000 000 000 016
   Las direcciones de los dos de long variable son cero porque mientras no ocupen espacio en la instancia sus apuntadores son nulos
   e) las longitudes de los dos atributis de long varible son cero por la misma razón anterior
   f) los valores iniciales de los atributos de longitud fija contienen ceros a largo de toda su longitud.
   g) los valores iniciales de los atributos de longitud variables no existen puesto que todavía no ocupan

ningún espacio dentro de la instancia.

Contenido de 0.i después de crear una instancia

```
0000000 000 000 000 000 000 000 000 002 000 000 002 000
0000014
```

El archivo de índices ahora ya contiene el apuntador hacia el primer bloque donde comienza la instancia # 1. La dirección de dicho bloque es 512_{10}.

4.4.1.2 La rutina que modifica atributos de los objetos realiza los siguientes pasos:

I. Obtiene del índice la dirección de la primer página del objeto dentro del archivo de datos.

II. Carga en memoria la descripción general de la clase que se encuentra en la página cero del archivo de datos (verifica antes si ya está en el buffer).

III. Carga en memoria la descripción particular de la instancia que se encuentra en la primera página del objeto.

IV. De la descripción general establece si el atributo es de longitud fija o variable. En el primer caso ir al paso V. En el segundo ir al paso VI.

V. Como es de longitud fija, hay que modificar el atributo sólo durante la longitud del mismo. Fin de la rutina.

VI. Si es variable hay que verificar si la nueva longitud que tendrá el atributo cabe dentro del espacio ocupado por el viejo atributo. Si cabe se realiza la modificación y termina. Si no cabe ir al paso VII.

VII. Si no cabe se abre el espacio necesario para lo que únicamente se tienen que recorrer los atributos de tamaño variable que estén a continuación del que se desea modificar.

VIII. Se reorganiza la tabla de apuntadores a los atributos de tamaño variable.

IX. Se actualizan los cambios en el disco.

La rutina para modificar desliza los demás atributos sólo cuando el nuevo valor no cabe. Cuando cabe y se necesita menos espacio lo deja desocupado. Esta forma de actuar obedece a modificaciones que se harán a la rutina en el futuro, como por ejemplo que al deslizar reaproveche los espacios ya no usados por algunos de los atributos previamente modificados.

A continuación se muestra un vaciado del archivo 0.d despés de modificar el primer atributo de longitud fija llenándolo con los siguientes diez caracteres: " color".

```
Contenido de 0.d después de modificar la instancia

0000000 000 000 000 000 000 004 000 003 000 002 000 001 000 004 000 005
0000020 000 002 000 003 000 001 000 003 000 003 000 001 000 002 000 012
0000040 000 004 000 004 000 000 000 000 000 000 000 000 000 000 000 000
0000060 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
*
0001000 000 000 000 000 000 000 002 044 000 000 000 000 000 000 000 012
0001020 000 000 000 016 000 000 000 000 000 000 000 000 000 000 000 000
0001040 000 000 000 000 040 040 040 040 040 143 157 154 157 162 000 000
0001060 000 000 000 000 000 000
0001066
```

4.4.2 La rutinas, de alto nivel, para accesar las asociaciones

Esta rutinas permiten dar de alta nuevas asociaciones entre instancias de objetos así como dar de baja o buscar asociaciones existentes. También permiten reccorer las asociaciones en forma secuencial ya sea desde al principio o desde la posición de una asociación especifica. Las asociaciones son almacenadas usando árboles B, con los que se simula un archivo "index sequential". Tal estructura es demasiado conocida por lo que ya no se trata aquí [BAYR72, WIRN76].

4.4.3 La rutinas, de bajo nivel, para accesar los archivos de los objetos

Estas rutinas realizan el manejo de "buffers" para los archivos de los objetos y además determinan el descriptor de archivo que se usará para las subsecuentes operaciones sobre un archivo. Esta propiedad es necesaria debido a que UNIX sólo permite tener abiertos 20 archivos como máximo, por lo que las rutinas para leer o escribir primero verifican si el

archivo ya está abierto, si lo está se realiza la operación usando el "buffer" que las rutinas asignan automáticamente, si no está abierto, las rutinas deciden cuál de los archivos y "buffers" limpiar (flush) y cerrar para reutilizarlo. El algoritmo para escoger "buffers" y descriptores de archivos se basa en el menos recientemente utilizado. Como en toda aplicación de bases de datos, el programa de aplicación debe iniciar la sesión de trabajo abriendo la base de datos y se debe despedir usando una rutina que se encarga de limpiar los "buffers" que aun esten sucios y que cierra todos los archivos antes de terminar la ejecución del programa. El propósito de esta rutina es garantizar que los datos de los archivos permanezcan en estado consistente.

Las rutinas para abrir archivos reciben a la entrada el nombre del archivo y el modo en el que se desea abrir (modo sólo lectura o lectura y escritura). Usando el nombre y el modo de operación del archivo se comprueba si este ya está abierto en ese modo, si no, se solicita que se abra y se procede a usarlo. Como se puede apreciar, las rutinas se encargan de garantizar que si un archivo ya está abierto en el modo solicitado, no se desperdiciará otro descriptor de archivo para abrirlo nuevamente. Esta característica tambien garantiza que todas las solicitudes de modificación a un archivo se harán sobre el archivo o "buffer" a él asignado.

4.4.4 La rutinas, de bajo nivel, para accesar los archivos de las asociaciones

Para usar una asociación, sólo se necesita pasar el nombre de la asociación a la rutina que se encarga de abrirlas, la cual regresa el descriptor interno de archivo (file descriptor en términos de UNIX), el cual se usa para los posteriores accesos al archivo de asociaciones. Las rutinas permiten tener varias asociaciones abiertas llevando el control de la posición de la última página accesada en cada una de ellas.

CAPITULO 5
DISCUSION Y CONCLUSIONES

5.1 Discusión sobre la aternativa de almacenamiento

Para escoger las alternativas de almacenamiento a usar en las BD/P y BD/AT se tuvieron en cuenta las necesidades y características de cada una de las bases de datos mencionas, así como las ventajas y desventajas expuestas en la sección anterior.

5.1.1 Discusión de la alternativa para la BD/P

Las características claves usadas para la selección de la alternativa de almacenamiento son su poder para representar redes de objetos y la rapidez para el acceso a los objetos componentes. La primera permite modelar mejor los objetos que ocurren en la realidad durante el proceso de diseño, la segunda es importante para lograr un sistema que resulte útil (de nada sirve un sistema en el que el diseñador tiene que esperar minutos u horas para que le responda).

La razón para no usar ninguna de las alternativas de tablas anidadas se encuentra en su falta de poder para representar redes de asociaciones, ya que únicamente pueden representar jerarquias de objetos. Sin embargo, resultan una opción muy tendadora por su conocida rapidez para recuperar objetos jerárquicos complejos del disco cuando estos se almacenan en bloques físicos cercanos, lo que minimiza el movimiento de las cabezas de los discos.

Quedan sin descartar las alternativas de tablas planas con relaciones binarias y n-arias, los defectos de ambas son los siguientes:

a. Las relaciones binarias presentan el problema de tener que usar una tabla diferente para asociar al mismo objeto con objetos de diferentes clases, con el consecuente problema de múltiples accesos a diversas tablas para recuperar los identificadores de los subobjetos que forman un objeto complejo.

b. Las relaciones n-arias para el manejo de asociaciones son poco atractivas porque tienen la desventaja de su rigidez para agregar asociaciones con nuevas clases de objetos diferentes a las inicialmente declaradas y de provocar el almacenamiento de valores nulos. Su ventaja es que eliminan la necesidad de entrar a múltiples tablas para recuperar los identificadores de los subobjetos de un objeto complejo.

c. Ambas opciones son más lentas que un sistema jerárquico o anidado.

Sin embargo, como modelar redes no es la única característica que se tiene que cumplir, sino que también se necesita que el sistema permita el acceso rápido a los objetos, se puso especial empeño en tratar de solucionar los problemas mencionados de las alternativas, y el único para el que no se encontró alguna alternativa adecuada fue para el de las relaciones n-arias. Debido a esto se escogió el esquema de tablas planas para representar los objetos y el uso de tablas binarias para representar las asociaciones entre aquellos.

Para resolver las deficiencias del manejo de múltiples tablas binarias se decidió dotar a cada instancia de objeto con un identificador substituto interno único o "surrogate"

[LORR83a] que indica la clase del objeto y el número de instancia dentro de esa clase. De esta manera las asociaciones ya no se realizan utilizando las llaves definidas por el usuario sino usando los identificadores sustituto. Esto se hizo con el propósito de poder usar una sola tabla por cada clase de asociación para eliminar la desventaja que se presenta, en cuanto al tiempo de acceso, por el uso de tablas diferentes para asociar cada pareja de clases de objetos, y cuya existencia era necesaria para que las aplicaciones de CAD pudieran diferenciar las distintas clases de los objetos entre los que existiera una asociación. Sólo fue posible unir todas esas tablas en una sola, que modele alguna asociación como la parte_de, al hacer que los identificadores indicaran la clase a la que pertenece cada objeto.

Al tener una sola tabla para cada clase de asociación ya no se tienen que hacer múltiples accesos a diferentes tablas para accesar todos los componentes de un objeto, dicho proceso queda reducido a hacer múltiples accesos a la misma tabla por lo cual queda totalmente descartado la alternativa de relaciones n-arias, ya que con esta modificación el esquema de relaciones binarias provee las mismas facilidades que el n-ario, sin presentar la desventaja de almacenar valores nulos.

Además, tener la tabla completa para una clase de asociación en un sólo archivo es importante en UNIX, porque cada operación para abrir o cerrar un archivo es muy cara en tiempo de ejecución y de accesos a los directorios del disco.

La única desventaja que persiste es tener los objetos en tablas separadas, lo que hace que la alternativa escogida sea más lenta que la jerárquica. Pero ese es el precio que se paga por tener ese poder de modelado superior. En la siguiente sección se propone cómo se podrá solucionar este problema usando técnicas de agrupamiento en disco y de "buffers" que se hacen necesarias debido a que el tiempo de respuesta resulta más critico en las BD/AT.

Solo resta por discutir algunas de las ventajas y desventajas del uso de identificadores internos sustitutos de las llaves primarias o "surrogates" [LORR83a]. Una de las ventajas es que ayuda a agrupar las asociaciones entre una clase de objeto determinada con otras clases de objetos diferentes bajo una sola asociación. El uso de los identificadores internos presenta el problema del uso de mayor espacio de almacenamiento para mantenerlos dentro del sistema. Lo que se gana es debido a la eliminación de los problemas derivados del uso de las llaves primarias definidas y controladas por el usuario. Tales problemas son: a) las llaves definidas por el usario son suceptibles de que se les necesite cambiar por lo que hay que repercutir el cambio a través de todos las entidades que las usen; b) algunas veces se necesitan usar diferentes llaves para identificar a la misma entidad; c) algunas veces es necesario almacenar información de una entidad, o tal vez sólo se quiera definir su existencia, pero el usuario no conoce o no tiene la llave.

Para ejemplificar el caso a) supóngase que se reliza un diseño de una planta en el que el identificador de las unidades de bombeo (bombas) tiene la forma B_n, donde n es un entero. Ahora supóngase que se necesita hacer un diseño

idéntico para una casa extranjera, para ellos el identificador deberá tener la forma P_n (porque bomba en inglés es Pump). Esto forza que se tengan que cambiar las llaves en el objeto donde se definen y en todos los lugares donde se les usen. Si se hubieran usado los "surrogates" para mantener las referencias, únicamente sería necesario hacer el cambio en los atributos del identificador de cada objeto y no sería necesario propagar el cambio a ningún otro lado porque el identificador interno nunca varía. La excepción a la regla se presenta en algunos casos particulares como cuando se necesita intercalar las informacionas de dos bases de datos o cuando se necesita recuperar del disco una instancia vieja que utilizó un identificador interno que ahora esté en uso por otra instancia. Por esta última razón nos inclinamos por evitar hasta lo máximo el reuso de los identificadores internos únicos.

El caso b) se puede dar en situaciones en las que se tienen dos llaves candidatas y para ciertas aplicaciones se requiere el uso de una y para otras el uso de la otra. Por ejemplo, en una compañía se lleva el control de los empleados por el número de empleado, pero si llega alguien que desea hacer alguna consulta acerca de alguno de los empleados difícilmente conocerá el número del empleado, lo que es más probable que conozca es el nombre y apellido del empleado.

El tercer caso podría darse cuando se está construyendo un nuevo auto o una nueva casa, pero aun no se le han marcado los números del registro federal de automóviles o aun no se conoce la numeración de la calle.

5.1.2 Discusión de la alternativa para las BD/AT

Se planteó utilizar, considerando algunas mejoras, la misma alternativa de organización usada en la BD/P para las bases de datos de las areas de trabajo (BD/AT). El motivo principal es que también en estas, como en la BD/P, se necesita poder modelar redes de asociaciones, aunque es muy cierto que se requiere que el tiempo de respuesta sea el mínimo posible. El problema de que este esquema (tablas planas con relaciones binarias) sea más lento que un sistema jerárquico o anidado se podrá resolver casi en su totalidad, si más adelante se usa un esquema de almacenamiento en el que en las BD/AT se agrupen aquellos objetos relacionados entre sí de manera que el almacenamiento físico (en el disco) se realice guiándose por la trayectoria más comúnmente usada por cada una de tales BD/AT, con lo cual se reduciría el tiempo de "seeking" en el disco; además será necesario usar un esquema apropiado de "buffers" para mantener en memoria las direcciones de los bloques de un objeto complejo y sus subobjetos, con lo que se logrará que la eficiencia de acceso sea similar a la de la organización jerárquica. Este sería, básicamente, el único cambio a realizar al sistema para que sea usado para llevar el almacemamiento en las BD/AT.

Al comparar esta solución con aquellas que comúnmente se usan para resolver el problema de lograr un esquema de alamacenamiento rápido, encontramos que estas últimas permiten almacenar los datos en forma de jerarquías agrupadas con base en el recorrido en preoden de la trayectoria con la que inicialmente se construyó el objeto. Lo cual significa que para agrupar en disco no se toma en

cuenta la frecuencia con la que son accesados ciertos subojetos u objetos asociados, por lo que no hay ninguna seguridad en cuanto a la eficiencia y rapidez de los accesos a los datos. Esta es la misma situación conocida que se da cuando se define una base de datos jerárquica, puesto que se está prefijando la forma de la jeraquia y del acceso, y no existe manera alguna en que la organización en disco vaya variando de acuerdo a los esquemas de uso de los datos, puesto que los datos estan fijos tanto a nivel de sus referencias lógicas como físicas.

Por otro lado, si se necesita modelar cualquier red, entonces hay que duplicar todos los datos involucrados en las dos o más jerarquias necesarias para representar dicha red, con los problemas para mantener actualizadas las copias y el uso innecesario de espacio.

Finalmente, otra caracteristica favorable de esta organización, es que permite accesar directamente cualquier objeto con sólo conocer su identificador y sin necesidad de leer y descifrar toda el área de control para acceso a la jerarqía, acción que es inevitable cuando se usa la organización jerárquica.

5.2 Evaluación del sistema

La solución planteada sirve para manejar el almacenamiento tanto en la base de datos del proyecto como en las bases de datos de las areas de trabajo. El sistema permite definir un máximo de 65K clases diferentes de objetos, 65K clases de asociaciones, cuatro giga instancias por clases e igual número de asociaciones entre las clases e instancias, por lo que prácticamente se puede decir que sólo está limitado por la capacidad del sistema operativo UNIX para manejar discos.

Por otro lado, el sistema permite trabajar con atributos de longitud variable o fija. Por supuesto, si se usan atributos de longitud variable se requiere más esfuerzo del sistema para llevar el control particular de cada atributo, lo cual no ocurre cuando son de longitud fija. Pero debido a las necesidades del ambiente de CAD y de otros muchos era necesario que se le incluyera la posibilidad de manejar los dos tipos de atributos.

Para realizar el diseño del sistema no sólo tomaron en cuenta los requerimientos originales mostrados en la primera sección del capítulo 3, sino que además permite:

- Que el sistema de almacenamiento pueda ser extendido muy fácilmente con nuevos procesadores especializados más adecuados para el proceso de tipos de datos especializados como imágenes por ejemplo (véase la sección 2.3). Esto significa que el sistema de almacenamiento es extendible, gracias a que se pueden declarar nuevos procesadores con sólo modificar un módulo en el que se declara el nuevo procesador. Para

realizar la e/s, el nuevo procesador podrá que utilizar las rutinas de bajo nivel de e/s, de otra manera el mismo se tendrá que manejar la e/s y proveer su propio esquema de manejo de "buffers" si lo necesita.

- Gracias a la forma en que se diseñó el manejo interno de los objetos, al utilizar una tabla directorio de bloques, el sistema permite almacenar atributos muy grandes en los que se necesite accesar solamente secciones específicas del atributo. Tal es el caso del acceso a cualquier elemento de arreglos (los arreglos son un tipo de datos muy usado en aplicaciones de CAD), también se puede hacer acceso directo a culquier parte de un texto sin tener que comenzar por el primer bloque de datos y leer todos los bloques intermedios antes de poder accesar el bloque que le interese a uno.

- Ralizando modificaciones muy pequeñas a las rutinas que manipulan los atributos, el sistema podrá manejar arreglos de tamaño fijo y variable.

Finalmente, debido a que UNIX sólo permite mantener abiertos un número pequeño de archivos, se incluyó al sistema el manejo automático de archivos y de "buffers", con lo cual decide automáticamente la asignación de descriptores de archivos libres o, en caso de no haber descriptores libres, cual de los archivos que se encuentran en uso en ese momento se puede cerrar para reasignar su correspondiente descriptor. Para este manejo se usó el esquema de el menos recientemente usado.

El hecho de tener desarrollado este trabajo, permite que pueda ser usado como marco de referencia para continuar con el desarrollo de otros módulos de la BD para CAD y con el mejoramiento de este mismo módulo. Por otro lado, el sistema de almacenamiento para objetos no sólo puede ser usado en el área de CAD, sino también como punto de partida en: automatización de oficinas, bases de datos convencionales, bases de datos estadisdicas lenguajes de programación e inteligencia artificial.

5.3 Posibles extensiones futuras

Para mejorar el sistema y su funcionamiento, por un lado se tendrá que definir completamente el diccionario de datos y, por otro lado, se deberá construir una interfaz propia de SALMO para realizar el acceso más eficiente, desde el punto de vista de objetos, al disco y se le podrá agregar el manejo de miniobjetos con el propósito de obtener mejor uso del espacio en disco.

La definición completa de DD queda totalmente fuera del proósito de este trabajo. En cambio a continuación se presentan los diseños de la interfaz de disco para SALMO, y el manejo de minibojetos.

5.3.1 La interfaz de SALMO al disco

Dicha interfaz será utilizada en lugar del sistema de archivos de UNIX, y unida a la parte que ya está desarrollada será llamada el sistema de acceso para objetos. Es decir, lo que se plantea es tener un sistema de archivos pero basado en objetos. Ya no se hará el acceso a estos a través de un sistema de archivos que no fue diseñado para sus características, lo que implica ganar rapidez y eficiencia en el manejo de los objetos.

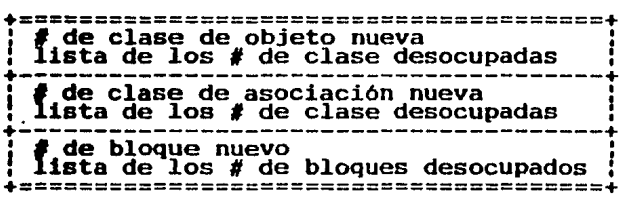
Esto implicará que ya no se podrá hacer uso de las utilerías que tiene UNIX para el manejo, verificación y mantenimiénto del sistema de archivos de UNIX, por lo que adicionalmente al la nueva parte del sistema para controlar el acceso al disco,se tendrán que construir también sus propias utilerias para hacer "dump", "restore", verificación y correcciones al sistema de archivos, las cuales deberán entender de la existencia de clases de objetos, clases de asociaciones y sus correspondientes instancias para realizar sus fucniones adecuadamente.

Esta reorganización del sistema seguirá sirviendo para manejar el almacenamiento en ambas bases dé datos, la del proyecto global y las de las areas de trabajo. En esta solución el sistema operativo UNIX sólo servirá para llevar el control del grupo de bloques asignados al nuevo sistema de archivos y que los accesos se hagan dentro de los límites físicos asignados al sistema de objetos.

El sistema de almacenamiento todavía podrá manejar las mismas cantidades de clases de objetos y asociaciones, y de instancias de unos y otros anteriormente descritas en el

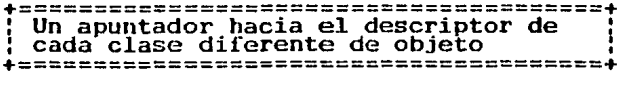
capítulo 4. El esquema general de organización siguirá conteniendo el descriptor general por clase de objeto y otro particular a cada instancia. Dado que ya no se usará el sistema de archivos de UNIX se tendrá que agregar algunas estructuras para poder crear y manejar las clases e instancias de objetos y de asociaciones. La organización de tales estructuras, las cuales harán las veces del sistema de archivos de UNIX, será la mostrada a continuación.

primer bloque del sistema de objetos



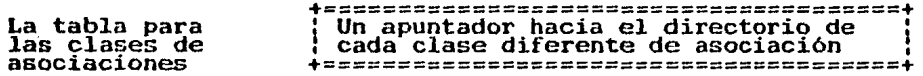
En este bloque se llevar' el control para asignar nuevos números de clases de objetos y asociaciones, asi como nuevos bloques para almacenar datos. El primer par de apuntadores se referirá a posiciones en la tabla de clases de objetos que se describe a continuación. El segundo par se referirá a la tabla de clases de asociaciones que se explica más adelante. Y finalmente el tercer par de apuntadores se referirá a posiciones del espacio restante del disco que se muestra al final.

La tabla para las clases de objetos



En esta tabla se tendrá una entrada por cada clase, en la que se almacenará el apuntador hacia lo que se ha denominado el descriptor directorio de la clase. Dicho

descriptor constará de dos tipos de información, el descriptor general de la clase y el directorio de la misma. El primero será semejante al expuesto en el capítulo 3, el segundo se explica adelante.



En esta otra tabla se tendrá una entrada por cada clase de asociación, en la que se almacenará el apuntador hacia el directorio de la clase. Dicho directorio permitirá accesar el árbol B para cada clase de asociación.

Este será el espacio restante del disco asignado al sistema de objetos. En el se contendrán: a) los descriptores directorio e índices de cada clase de objetos; b) los directorio de cada clase de asociacion; c) los datos de las instancias de los objetos y sus asociaciones.

Cada uno de los descriptores directorio que son apuntados desde la tabla para clases de objetos está constituido de la siguiente forma:

a. Contendrá el descriptor general para esta clase en el que se determinarán los atributos, sus tipos y

longitudes si es el caso.

b. El número de instancia nueva que se podrá asignar en esta clase.

c. La lista de los números de instancias desocupadas en esta clase

d. El número de bloques que tiene la tabla directorio de esta clase.

e. El número de bloques que tiene la tabla índice de esta clase.

f. Este apuntador tendrá varios usos dependiendiendo de los valores de los tres campos anteriores. Si el número de instancia nueva es menor a 128 apuntará al primer y único bloque de la tabla incice. Si el número de bloques de la tabla índice es menor de 128 apuntaá al primer y único bloque de la tabla directorio. Finalmente si no es ninguno de los dos casos anteriores, apuntará al primer bloque de la lista ligada, de bloques, que contendrá los apuntadores a los bloques usados por la tabla directorio.

g. La tabla directorio que contendrá los bloques usados por la tabla de índices.

h. La tabla índice que contendrá los apuntadores al primer bloque de cada instancia.

Gracias a la estrategia de uso diferente del apuntador del punto (f) se tendrá que para las primeras 128 instancias sólo existirá un nivel de indirección para el acceso a los datos. Cuando el número de instancias aumenta a más de 129 y hasta 2^{14} se realizan dos indirecciones. Y después de ese punto se tienen que realizar tres indirecciones. Lo anterior solamente es reflejo de que el diseño del sistema de objetos se realizó para que tenga su máxima eficiencia cuando existan pocas instancias de cada clase, es decir que es más eficiente cuando se maneje una variedad más rica de clases de objetos.

Por otro lado cabe destacar que aun en el peor de los casos, en el que se manejara un exesivo número de instancias de cada clase, el sistema sería casi el doble de eficiente que si se usara el sistema de archivos de UNIX.

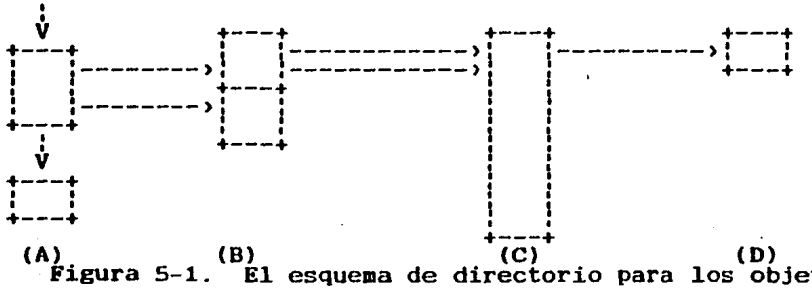


Figura 5-1. El esquema de directorio para los objetos

En la figura 5-1 (A) será la lista de bloques de la tabla directorio; (B) la tabla directorio en la que se encuentrará la lista de bloques del índice; (C) la tabla índice desde la que se apuntará a las instancias; y finalmente (D) es el espacio usado por las instancias.

Los directorios para las asociaciones mantendrám el bloque descriptor desde donde se apuntará a la raiz del árbol B, asi como a los bloques inicial y final de las hojas del árbol.

La organización que tendrán las bases de datos de las áreas de trabajo será la misma ya planteada, sólo que se pondrá especial atención en utilizar bloques contiguos de disco, minimizando asi el tiempo de "seek", para almacenar los objetos, sus asociaciones y datos de control en base a la trayectoria más frecuentemente usada por esa BD/AT al accesar los datos de un proyecto. Además se podria decidir que el tamaño de los bloques en estas áreas de trabajo sea de longitud variable. Ambas acciones estarán orientadas a conseguir que el tiempo de acceso a los datos en disco, basados en la trayectoria especifica más usada, sea más rápido que en la base de datos del proyecto, pero conservando el poder para modelar las redes de asociaciones diferentes entre los objetos.

5.3.2 El manejo de miniobjetos en SALMO

Se presentan las modificaciones para el mejor aprovechamiento del espacio en disco cuando se manejen instancias de objetos muy pequeñas.

Se ideó la noción de objetos minúsculos con el próposito de mejorar el aprovechamiento del espacio en disco. Las reglas de su manejo se listan a continuación.

a) Si al crear un objeto tiene tamaño < (B_TAM / 2), entonces se manejará como objeto minúsculo o miniobjeto, esto es, se guardará agrupando varios objetos en cada bloque.

b) Si al intentar crecer alguno de los atributos de un miniobjeto se rebasa el espacio disponible en un bloque, entonces se moverá el objeto infractor a un bloque nuevo y se regresará el espacio desocupado por el infractor a la lista de segmentos libres en dicho bloque. El objeto infractor continuará siendo tratado como miniobjeto mientras su tamaño sea menor que el indicado en el punto a). En caso contrario se deberá actualizar la bandera que hay en su entrada correspondiente en el archivo índice.

c) Los miniobjetos sólo podrám coexistir con otros miniobjetos en el mismo bloque pero nunca con objetos que no sean minúsculos.

Para llevar a cabo el manejo de miniobjetos se tendrán que hacer los siguientes cambios u adiciones. A la página cero de la clase se le agregara un apuntador de tres bytes que indicará el primer bloque de la lista de bloques que conteniendo miniobjetos aún tengan espacio libre.

Los últimos dos bytes de aquellos bloques que contengan instancias de miniobjetos, se usarán para apuntar al primer segmento de la lista de segmentos libres dentro de ese bloque, asi como para contener una bandera que indique si el espacio dentro de este bloque de miniobjetos necesitará reorganización debido a fragmentación de su espacio libre. El siguiente byte indicará el tamaño del arreglo de parejas que permitirá conocer las posiciones de los miniobjetos

dentro de este bloque. El arreglo de parejas mencionado estará organizado de la siguiente manera ‹número de instancia, apuntador (^) al primer byte de esa instancia dentro de este bloque›. Esta organización está diseñada con el propósito adicional de que las instancias dentro de los bloques no dependan de su posición física.

Entrada en el archivo de índices hacia el archivo de datos

^ bloque con la instancia	bandera miniobjeto

Bloque conteniendo miniobjetos dentro del archivo de datos

espacio para los datos de algunos de los diversos miniobjetos		
que se encuentran en este bloque	espacio libre	espacio
para otros miniobjetos	más espacio libre, etc ...	
arreglo de ‹# instancia, ^ 1er byte local›		tamaño arreglo
^ 1er segmento		

Al crear una instancia minúscula se prenderá una bandera en el apuntador desde el archivo indice que indica que es un miniobjeto, por lo tanto al tratar de hacer cualquier acceso o modificación a esa instancia sabrá que los datos estarán organizados como miniobjetos dentro de ese bloque. Por otro lado se tendrá un apuntador desde la página cero de la clase que permitirá llegar a todos aquellos bloques que contengan instancias minúsculas y bloques libres.

En el archivo de los datos de los objetos el espacio disponible dentro de cada bloque estará manejado por una lista de segmentos libres. Cada segmento usará sus primeros dos bytes para apuntar al siguiente segmento y los siguientes dos bytes para indicar el tamaño del propio

segmento. El tamaño mínimo permitido para formar un segmento será el que sea mayor de entre el tamaño que tiene una instancia de esa clase de objeto recién creada o dieciseis bytes. Por lo tanto el tmaño mínimo minimorum asegurado es de dieciseis bytes.

Además se usarán los primeros tres bytes del primer segmento libre de cada bloque para apuntar al siguiente bloque con miniobjetos y espacio disponible.

NOTA: Esta idea es totalmente diferente de lo que el Dr Buchmann había establecido para el manejo de objetos, ya que a él sólo le interesaba poder manejar objetos de tamaño grande ilimitado y su idea era que cada objeto usara un bloque de disco mínimo debido a que el número de instancia del objeto era el número de bloque de disco, y los bloques adicionales se encontrarían en otro archivo nombrado archivo de sobreflujo o desborde. Este manejo de un objeto por bloque se debe en parte a que el propósito del sistema es almacenar objetos de diseño de plantas químicas que generalmente tienden a ser grandes por lo que no se apreciaba que fuera necesario cargar con el sobrecosto de mantener instancias muy pequeñas. En alguna etapa posterior se puede agregar este caso de miniobjetos si se encuentra que es necesario para el manejo de CAD en plantas químicas o si se usa el sistema en alguna otra aplicación.

Ahora que se han descrito las posibles mejoras que se pueden hacer para el almacenamiento, en general, de objetos y sus asociaciones, sólo resta decir que aun después que se realizen estos cambios podría pensarse en hacer algunas

otras mejoras para almacenar mayor cantidad de información en menos bloques de disco. Para lograr esto se plantea la posibilidad de utilizar algún algoritmo de compactación tales como el uso de prefijos o postfijos en las llaves del árbol B o también buscar la forma de compactar los atributos de los objetos, especialmente los atributos de longitud variable. Sin embargo, estas decisiones hay que valorarlas detenidamente puesto que este sistema, como cualquier otro, no se escapa de la ley no escrita de la computación que determina que a mayor complejidad de los datos hay mayor costo para el proceso de los mismos.

BIBLIOGRAFIA

[BALR83]

Balza, R. M., et al, Data base technology applied to engineering data, Proceedings of the second intnal conference on foundations of computer-aided process design, Snowmass, Colorado, 1983.

Describe los requirimientos impuestos a los sistemas manejadores de bases de datos para que resulten apropiados para soportar ambientes de diseño. Este trabajo presenta el proyecto que desarrolló la Boing para la NASA con el fin de realizar un conjunto de programas integrados para el diseño de veiculos aeroespaciales (IPAD por sus siglas en inglés).

[BATD84]

Batory, D.S, A.P., Buchmann, "Molecular Objects, Abstract Data Types, and Data Models: A Framework", 10th International conference on Very Large Data Bases, Singapure, Aug 1984.

Se estudia el concepto de Objeto Molecular, y se propone el modelo de tipos de datos abstractos para modelar objetos.

[BUCA80]

Buchmann, A. P., A methodology for logical design of databases for proyect engineering, Tesis doctoral, Universidad de Austin Texas, 1980.

[BAYR72]

Bayer, R, E, McCreight, "Organizational and Maintenance of large ordered indexes", Acta Informática, 1, Num. 3, 1972.

[BUCA85]

Buchmann, A.P., C., Perez de Celis; "An arquitecture and data model for CAD Databases", Proc. 11th Intl. Conf on Very Large Data Bases, Stockholm, Aug 1985.

BIBLIOGRAFIA

Se describe el modelo de una BD orientada a CAD, asi como las funciones de cada una de sus partes.

[BUCA86a]
Buchmann, A.P., R.S., Carrera, M.A., Vázquez, "A generalized Constraint an Exception handler for an Object-Oriented CAD-DBMS", Proc. 2nd Intl workshop on Object Oriented Data Base Systems, Asilomar, California, EE. UU. Sep 86.

Se presenta un manejador especializado para restricciones y excepciones, estableciendo su importancia dentro de un SMBD para CAD.

[BUCA86b]
Buchmann, A. P., R. S., Carrera, "El uso de un sistema convencional de CAD versus un manejador de bases de datos especialmente diseñado para manipular datos de CAD", memorias de la II reunión nacional de CAD/CAM, Cuernavaca, Morelos, México, Nov 86.

Contiene la descripción general del SMBD para CAD estableciendo las caracteristicas mas importantes que debe poseer dicho manejador.

[CARR85]
Carrera-S., R. S., Noe, Hernandez, Bases de datos estadisticas, principales problemas y posibles soluciones, IIMAS serie monografias núm. 85, 1985.

[CODA71]
"CODASYL data base task group april 71 report", ACM, Nueva York, Nueva York, 1971.

[CODE70]

BIBLIOGRAFIA

Codd, E. F., "A relational model for large shared data", Communications of the ACM, volumen 13, numero 6, junio de 1970, páginas 377-387.

[CODE79]
Codd, E. F., "Extending the database relational model to capture more meaning", Proceedings of the ACM Transactions on database systems, Vol. 4, Num. 4, 1979.

[DAWF82]
Dawson, F. T., CAD in plant design: a company viewpoint, en Computer aided process plant design, editor M. Leesley, 1982.

[GOLA83]
Goldberg, A., D., Robson, SMALLTALK-80 the language and its implementation, Addison-Wesley, 1983.

[GUTA82]
Guttman, A., M., Stonebraker, "Using a relational database management system for computer aided design data", Data Base Engineering, Vol 5, Num. 2, junio de 1982

[HASR82]
Haskin, R. L., R. A., Lorie, "On Extending the functions of a relational database system", Proceedings of the ACM SIGMOD international conference on Management of data, 1982.

[JOHH83]
Johnson, H.R., et. al., "A DBMS facility for handling structured engineering entities", Database week, San José Calif., 1983

BIBLIOGRAFIA

[KORH86]
Korth, H. F., A.,Silverschatz, Database System concepts, McGraw Hill, 1986.

Se muestran los diferentes modelos de datos que se usan ordinariamente en el área de las bases de datos y los nuevos modelos usados en las nuevas clases de aplicaciones que han surgido.

[LEEM82]
Leesley, M., E., . Editor, Computer-aided process plant design, Gulf publishing company, 1982.

Es un libro formado por una serie de articulos invitados que tratan el problema de CAD para plantas de diseño. Van desde el análisis del tipo de personal para manejar los sistemas de CAD, pasando por algunas soluciones particulares y llegando hasta establecer cuales serán las tendencias futuras en el área.

[LOHG83]
Lohman, G. M., et al., "Remotely sensed geophysical databases: Experience and implications for generalized DBMS", Proceedings of the ACM SIGMOD international conference on Management of data, 1983.

[LORR83]
Lorie, R., W., Plouffe, "Relational databases for engineering data", Proceedings of the second international conference on Foundations of Computer aided process design, Snowmass, Colorado, E. U., junio de 1983.

Propone el uso de sistemas de bases de datos relacionales para llevar el manejo de los datos de la ingenieria de diseño. Reconoce las limitaciones del modelo y propone extensiones al mismo para que resulte apropiado para este

uso.

[LORR83a]
Lorie, R., W., Plouffe, "Complex objects and their use in design transactions", Database week, San José, California, mayo de 1983.

[PROS83]
Proctor, S. I., " Challenges and constraints in computer implementation and, applications", Proceedings of the second international conference on Foundations of Computer aided process design, Snowmass, Colorado, E. U., junio de 1983.

[SMIJ77]
Smith, J. M., D. C. P., Smith, "Database abstractions: aggregation and generalization", ACM Transactions on database systems, julio, 1977.

[STOM83]
Stonebreaker, M., et al, "Application of abstract data types and abstract indices to CAD databases", Proceedings on Engineering Design and Applications of ACM-IEEE database week, San José, Ca., mayo de 1983.

[STOM86]
Stonebreaker, M., "Object management in POSTGRES using procedures", Proceedings of International workshop on OODBS, Pacific Grove, California, 1986.

[TZID82]
Tsicritzis, D.C., F.H., Lochovsky, data models, Prentice Hall, 1982.
Explican los conceptos de modelos de datos para bases de

datos asi como su estructura, restricciones y operaciones, cubriendo siete modelos distintos.

[ULLJ80]
Ullman, J.D., Principles of Database Systems, Computer Science Press, 1980.

Se presentan diferentes organizaciones físicas para bases de datos asi como la teoria sobre la que se basa, principalmente, el modelo relacional.

[UMED85]
Umeshwar, D, et al, PROBE- A Research Project in Knoledge-Oriented Database Systems: Preliminary Analysis, Technical Report CCA-85-03, Computer Corporation of America, 1985

Presenta el estudio de las caracteristicas que deberán tener los nuevos SMBD para resolver aplicaciones en las que resulta necesario el uso intensivo de conocimientos, como en el caso de la automatización de negocios, oficinas, industrias, CAD/CAM, control y órdenes militares y cartografía.

[WINP83]
Winter, P, C.j., Angus, "The database frontier in process design", Proceedings of the second international conference on Foundations of Computer aided process design, Snowmass, Colorado, E. U., junio de 1983.

[WIRN76]
Wirth, N, Algorithms + data structures = programs, Prentice Hall, 1976.