58
2y

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ARAGON"

INGENIERIA EN COMPUTACION

BASES DE DATOS ORIENTADAS A OBJETOS

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE :

INGENIERO EN COMPUTACION

PRESENTA:

PEDRO RIVERA AVALOS

ASESOR DE TESIS:

ING. ERNESTO PEÑALOZA ROMERO

SAN JUAN DE ARAGON, ESTADO DE MEXICO, 1996

TESIS CON
FALLA DE ORIGEN

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGON

Tema:
BASES DE DATOS ORIENTADAS A OBJETOS

Objetivos del Tema:

- Proporcionar un panorama de las bases de datos Orientadas a Objetos.
- Proporcionar una **introducción a la terminología** de la orientación a objetos.
- Mostrar algunas ventajas y desventajas de las bases de datos orientadas a objetos, comparándolas con las relacionales.
- Hacer una proyección corta del futuro inmediato de las bases de datos orientadas a objetos.

Capítulos por Desarrollar:

INTRODUCCION

CAPITULO I. ANTECEDENTES HISTORICOS DE LAS BASES DE DATOS.

CAPITULO II. INTRODUCCION A LA FILOSOFIA Y TERMINOLOGIA DE LA ORIENTACION A OBJETOS.

CAPITULO III. BASES DE DATOS ORIENTADAS A OBJETOS (BDOO).

CAPITULO IV. PERSPECTIVAS FUTURAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS (BDOO).

CONCLUSIONES

GLOSARIO

BIBLIOGRAFIA

Asesores:

Ing. Ernesto Peñaloza Romero

Ing. Gabriela González Hernández

Ing. Silvia Vega Muytoy

Ing. Silvestre López Abundio

Ing. Martín Ordóñez Rosales

Iba un hombre caminando por el desierto cuando oyó una voz que dijo: "Levanta algunos guijarros, mételos en tu bolsillo y mañana te sentirás a la vez triste y contento."

Aquel hombre obedeció. Se inclinó, recogió un puñado de guijarros y se los metió en el bolsillo.

A la mañana siguiente, vio que los guijarros se habían convertido en diamantes, rubíes y esmeraldas. Y se sintió feliz y triste.

Feliz por haber recogido guijarros; triste por no haber recogido más.

Lo mismo ocurre con la educación.

W. CUNNINGHAM

Escribir no es fácil, implica sacrificio tanto para el que lo hace, como para los que le rodean. En agradecimiento por este hecho dedico esta obra a:

Dios:
Por haberme dado la grandeza de mi familia y mi profesión, gracias señor...

Mi madre
Elisa Avalos González
Que me da amor, cariño y ternura, y me enseñó el camino que ahora sigo. Espero que al terminar este trabajo haya cumplido uno de sus más grandes anhelos.

Mi esposa
Araceli Pérez Santana
Con todo mi amor para quien con amor, paciencia y confianza, recorre mi camino y de quien he aprendido mucho. Sé que tú también lo lograrás, cuenta conmigo.

Mis hijos:
Kevin Abraham y Lluvia Itzel Rivera Pérez
A ustedes que son parte de mí, ésto es un incentivo para que inicien una de las realizaciones más nobles y satisfactorias en la vida: Una profesión. Recuerden ¡no existe el tiempo, ni el espacio, entre ustedes y yo; siempre estaré con ustedes. Esfuércense y logren sus metas, no se limiten..., y recorran sus caminos!.

Mi hermana y mi cuñado:
Elba Rivera Avalos y Jorge Llaguno Millán
Gracias por su ayuda, consejos y cariño.

Mis tíos
Flora Rivera Pérez, Edmundo Rivera Pérez, María Rivera Pérez y Rosario Avalos González.
Gracias por su ayuda desinteresada, que desde niño me brindaron para que yo pudiera lograr esta carrera. Este es un testimonio de mi cariño hacia ustedes.

A la memoria de mi abuelo
Pedro Rivera Nava,
a mis abuelos
Elpidia Pérez, Alfredo Avalos y Enedina González
Símbolos de bondad, rectitud y cariño.

A mi padre
Pedro Rivera Pérez, gracias de alguna forma has contribuido para que yo esté aquí.

A LA UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

A LA FACULTAD DE INGENIERIA

A LA ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ARAGON"

A mis maestros
Con reconocimiento y gratitud.

A mis asesores
Por el apoyo que me brindaron para la estructuración de esta obra.

Y a todos mis familiares y amigos:
Que de algún modo u otro me ayudaron y motivaron durante todos estos años, cuyo ánimo y apoyo a lo largo de la realización de este trabajo, me han hecho ver a este como algo importante, útil y necesario.

BASES DE DATOS ORIENTADAS A OBJETOS

INDICE

pag.

INTRODUCCION 1

CAPITULO I.- ANTECEDENTES HISTORICOS DE LAS BASES DE DATOS.

1.1.- LA CRISIS DEL SOFTWARE 6
1.2.- LOS SISTEMAS DE ARCHIVOS PLANOS 13
1.3.- ALMACENANDO DATOS EN UNA BASE DE DATOS 15
1.4.- DBMS MANEJADORES DE BASE DE DATOS 19
1.5.- LA EVOLUCION DE LAS ESTRUCTURAS DE LAS B. D 23
1.6.- OTROS CONCEPTOS REFERENTES A LAS BASES DE DATOS 39
1.7.- EL CICLO DE VIDA DE UN SISTEMA 41
1.8.- NUEVAS FRONTERAS PARA APLICACIONES DE B.D 44
1.9.- ARQUITECTURAS DE LAS BASES DE DATOS 46

CAPITULO II.- INTRODUCCION A LA FILOSOFIA Y LA TERMINOLOGIA DE LA ORIENTACION A OBJETOS.

2.1.- LA FILOSOFIA DE LA ORIENTACION A OBJETOS 51
2.2.- OBJETOS 58
2.3.- CLASES E INSTANCIAS 69
2.4.- HERENCIA Y GESTALT 75
2.5.- JERARQUIAS DE CLASES 78
2.6.- ABSTRACCION 81
2.7.- ENCAPSULACION 83
2.8.- LA COMPLEJIDAD DE LA ESTRUCTURA 84
2.9.- SOBRE CARGA DE OPERADORES 85
2.10.- LIGADURA DINAMICA 86
2.11.- BLOB (BNARY LARGE OBJECT) 88
2.12.- PROGRAMACION VISUAL 89
2.13.- LENGUAJES DE PROGRAMACION ORIENTADOS A OBJETOS LPOO 90
2.14.- UN MODELO CONCEPTUAL UNIFICADO 94
2.15.- RELACION COSTO / BENEFICIO DE LA TECNOLOGIA DE OO 101

pag.

CAPITULO III.- BASES DE DATOS ORIENTADAS A OBJETOS.

3.1.- LA EVOLUCION DE LOS DBMSOO 105
3.2.- FUNCIONALIDAD DE LAS BASES DE DATOS OO 115
3.3.- LOS ORIGENES DE LOS DBMSOO 116
3.4.- CONCIBIENDO UN MEJOR DBMS 118
3.5.- PERSISTENCIA PARA LOS LENGUAJES DE PROGRAMACION OO 120
3.6.- CONTENEDORES DE COMPONENTES DE SOFTWARE 123
3.7.- ALMACENAMIENTO DE OBJETOS COMO ELEMENTOS DE DATOS 125
3.8.- TECNICAS PARA CONSTRUIR UN DBMSOO 138
3.9.- COMPONENTES DE UN OODBMS 147
3.10.- ENTENDIENDO EL DISEÑO DE UN DBMSOO 155
3.11.- TECNICAS DE LOS DBMSOO PARA EL MANEJO DE LOS METODOS 158
3.12.- ADMINISTRACIÓN DE SEGURIDAD EN LAS TRANSACCIONES 162
3.13.- BASES DE DATOS ORIENTADAS A OBJETOS DISTRIBUIDAS 165
3.14.- SOPORTE PARA WORKGROUPS 167
3.15.- VERSIONES DE OBJETOS 168
3.16.- DESARROLLO CON BDOO 169
3.17.- PROBLEMÁTICA EN LAS BDOO 171
3.18.- LENGUAJES DE CONSULTA AD-HOC 176
3.19.- INTEGRIDAD DE LOS DATOS 180
3.20.- BASES DE DATOS RELACIONALES VS BDOO 184
3.21.- LOS BENEFICIOS DE LOS DBMSOO 189

CAPITULO IV.- PERSPECTIVAS FUTURAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS (BDOO).

4.1.- EL TIEMPO DEL CAMBIO ES AHORA 193
4.2.- EL SUBENIR DE DBMSOO 195
4.3.- TECNOLOGIA INFORMATICA DEL SIGLO XXI 210
4.4.- TEMAS TECNICOS EN EVOLUCION 212
4.5.- EMPRESAS QUE YA ESTAN UTILIZANDO LA TECNOLOGIA 216

CONCLUSIONES 218

GLOSARIO 221

BIBLIOGRAFIA 237

INTRODUCCION

INTRODUCCION

La concepción del mundo es cada vez más y más compleja. Cada día nueva información se suma al acervo existente del conocimiento. Se estima que la cantidad de información en todas las áreas (tecnología, ciencia, comercio, historia, etcétera) del mundo se duplica cada cinco años. Este enorme acervo de conocimiento hace imposible que una sola persona lo adquiera, aún se trate de entender únicamente pequeñas y específicas áreas del universo. La búsqueda de información acerca de temas relevantes está siendo cada vez más difícil de tratar. Las empresas de hoy en día cada vez se están convirtiendo en organizaciones basadas en información; sin embargo, su habilidad para manejar esos datos se ve disminuida porque el volumen de información se expande más rápido que su capacidad de procesarla, el resultado de ello, es que las empresas prácticamente se están ahogando en sus propios datos.

Una considerable cantidad de los procesos actuales del desarrollo de software continúa con su carácter preindustrial y sus programadores funcionan todavía como artesanos. Crean componentes únicos, no intercambiables y los unen manualmente; a continuación emplean una enorme cantidad de tiempo extra en comprender el código creado por sus predecesores, ampliar y refinar el programa. Después de casi cincuenta años de programación de computadoras, se ha dado el siguiente paso en la exploración de nuevas y mas eficientes técnicas del desarrollo de sistemas: el paradigma de la Orientación a Objetos (OO). La idea de estas nuevas herramientas es reemplazar la programación mediante el armado o ensamble lógico de elementos predefinidos. Estos elementos se ensamblan regularmente mediante un ambiente gráfico. El orden y la posición donde se coloquen, definen la lógica de los requerimientos de la empresa, a estos elementos se les conoce como objetos. Durante los 80s los DBMS Relacionales han predominado en el mercado, al mismo tiempo las aplicaciones han evolucionado: CAD/CAM, Automatización de oficinas, Ingeniería de Software, multimedia, diseño VLSI, etc. Nuevos tipos de datos han surgido: imágenes, voz, gráficas, programas, etc.; los DBMS relacionales no han

podido soportar las aplicaciones llamadas no estándar, lo que ha dado lugar a numerosos estudios para desarrollar una mejor base de datos que soporte aplicaciones avanzadas, esta búsqueda ha dado lugar a dos diferentes categorías:

- **La evolución de la tecnología**: Con este enfoque el modelo relacional ha ido evolucionando con extensiones y adaptaciones para soportar objetos
- **La revolución de la tecnología**: Basada en un nuevo modelo de datos el modelo orientado a objetos

FINALIDAD

- Proporcionar un panorama de las bases de datos Orientadas a Objetos;
- Proporcionar una introducción a la terminología y la filosofia de la Orientación a Objetos;
- Mostrar algunas ventajas y desventajas de las bases de datos orientadas a objetos (bases de objetos), comparandolas con las relacionales;
- Hacer una proyección corta del futuro inmediato de las bases de datos orientadas a objetos;

El presente trabajo de Tesis se integra de cuatro capítulos, conclusiones, un glosario de términos y la bibliografía. A continuación se describe brevemente su contenido:

Capítulo I. Antecedentes Históricos de las Bases de Datos. Se presenta un panorama general acerca de la crisis del software, en él se menciona cómo en la actualidad, la mayoría del software corporativo es obsoleto antes de liberarse y frecuentemente incapaz de evolucionar para satisfacer necesidades futuras. Se muestra también cómo ha evolucionado el término bases de datos junto con el de modelo de datos. Se describe brevemente lo que es un DBMS, para lo cual se definen algunos conceptos, entre ellos: sistema, sistema de base de datos, y base de datos distribuída. Por ultimo se analiza de forma general los diferentes problemas de integración de las bases de datos, así como la diferente arquitectura para dicha integración.

Capítulo II. Introducción a la Filosofía y la Terminología de la Orientación a Objetos. Se presenta una introducción clara y concisa de la flosofía, términos y técnicas de la orientacion a objetos, definiendo con ejemplos los conceptos fundamentales del paradigma de la orientación a objetos, como son: objetos, clases, estructura y encapsulamiento, mensajes y protocolos, métodos, jerarquías de clases, herencia y polimorfismo, lenguajes de programación; y al final se analiza la relación costo-beneficio de la tecnología.

Capítulo III. Bases de Datos Orientadas a Objetos (BDOO). Es la parte central del presente trabajo. Se analizan las doce reglas de oro que definen un DBMSOO, se define el concepto de evolución (Bases de datos Relacionales Ampliadas o Extendidas) y revolución (Bases de datos Orientadas a Objetos Puros) de la tecnología de base de datos. Se describen algunas de las ventajas y desventajas más representativas de las bases de datos orientadas a objetos (puras) comparándolas para ello con las bases de datos relacionales (híbridas); una vez diferenciándolas, se analizan sólo las puras, excluyendo de esta tesis las bases de datos relacionales extendidas (híbridas). Se muestra cómo el problema del manejo de objetos complejos es manejado por los DBMSOO por medio de IDs (Identificadores Unicos) que permiten la persistencia de los objetos. También se analiza la arquitectura de un DBMSOO, concluyendo mencionando los beneficios más representativos de los DBMSOO.

Capítulo IV. Perspectivas Futuras de las Bases de Datos Orientadas a Objetos (BDOO). Se expone acerca de las arquitecturas orientadas a objetos en los 90s, se comentan los temas técnicos de las BDOO que se encuentran en evolución; se analizan los OODBMS existentes en el mercado y se efectúa en base a los capítulos anteriores, una proyección personal referente al futuro próximo de las Bases de Datos Orientadas a Objetos. Se muestra brevemente, lo que es una base de datos inteligente. Se mencionan también algunos ejemplos de empresas que han migrando sus sistemas de bases de datos hacia la orientación a objetos y los resultados que han obtenido.

Conclusiones. Se concretan y resumen de forma general los aspectos más importantes de esta tesis.

Glosario. Se definen algunos términos y siglas que se emplean en esta tesis. La definición completa de los términos se puede encontrar en el cuerpo principal de la tesis. El acrónimo OO para orientado a objetos, BD para base de datos y DBMS para Data Base Management System (Manejador de Base de Datos) aparece en todo el trabajo.

MAGNITUD Y TRASCENDENCIA

La magnitud y trascendencia del presente trabajo de tesis esta sustentada básicamente en dos atributos:

I.- Es un material que responde a una necesidad de servicio a la comunidad. Como parte del desarrollo histórico de la enseñanza en México, está sin lugar a duda la incorporación de la educación universitaria al servicio de la comunidad. Propuesta de servicio que la Universidad Nacional Autónoma de México consideró desde sus inicios, e incluso hasta nuestros días; prueba de ello lo constituye la especificación y actualización de sus planes de estudio, posibilitando la integración mas temprana de la juventud al sector productivo de México.

II.- La base conceptual que es abordada, da vistas de actualidad y de aplicación en nuestro medio. Uno de los más grandes problemas de los países en vías de desarrollo, como son los de América Latina, es su dependencia económica y tecnológica de los países desarrollados. Por tal motivo, no contar con buena infraestructura para desarrollar aplicaciones productivas con los recursos que contamos en nuestro país o desconocer la existencia de éstos, nos limitan o atrasan en la utilización de nuevas herramientas de software o de técnicas para el análisis y desarrollo de sistemas computacionales. Aceptando el calificativo de ser una propuesta, este trabajo es "perfectible", e incluso cuestionable bajo otros enfoques; sin embargo, fue mi decisión realizarla en espera de favorecer las labores de investigación de algún otro compañero que desee profundizar o teorizar acerca del tema propuesto en la presente Tesis.

La aplicación de los citados atributos al trabajo de investigación y desarrollo del tema expuesto, será especificada en páginas subsecuentes, quedando a consideración su lectura.

ALCANCES Y LIMITACIONES

Ya que con este trabajo se pretende brindar al lector su primer panorama general de las bases de datos orientadas a objetos, determinaré el alcance y las limitaciones para satisfacer dicho objetivo.

Es importante aclarar que a diferencia del modelo relacional que está sustentado sobre un modelo matemático estándar propuesto por F. Cood, el modelo orientado a objetos no tiene un modelo estándar, ya que existen muchos modelos que cada fabricante de DBMSOO formaliza y propone, y aplica sus teorías; sin embargo, todos ellos parten de los mismos conceptos, brindando las mismas bondades. Es preciso citar que el enfoque para el desarrollo de esta tesis es totalmente teórico, debido a que me fue imposible conseguir el producto, por lo que esta investigación se basa gran parte en los conocimientos obtenidos en el Simposium de Ingeniería de Software impartido por la Fundación Arturo Rosenblueth, muy particularmente en la conferencia de BASES DE DATOS ORIENTADAS A OBJETOS. Además, del conocimiento que nos brindan los libros y documentos encontrados para la elaboración de este tema, se encontraron manuales de referencia del DBMSOO Versant.

CAPITULO I

ANTECEDENTES HISTORICOS DE LAS BASES DE DATOS

Los DBMS Relacionales son realmente para datos de registros. Pero la mayor parte del mundo real no está hecho de registros.

Eugene Wong (Noviembre 1988).

CAPITULO I

ANTECEDENTES HISTORICOS DE LAS BASES DE DATOS

1.1 LA CRISIS DEL SOFTWARE

Al inicio de los años 50, cuando en el mundo solo había seis computadoras, el reducido número de programadores que existían era el cuello de botella que impedía explotar al máximo estas máquinas. En la actualidad, el número de computadoras alcanza los cien millones con una velocidad y potencia impresionante, pero el software sigue siendo el cuello de botella; desarrollar el software que iguale el potencial de las computadoras, resulta ser un reto mucho mayor que el construir máquinas mucho más rápidas. Una considerable cantidad de los procesos actuales del desarrollo de software continúa con su carácter preindustrial y sus programadores funcionan todavía como artesanos. Crean componentes únicos, no intercambiables y los unen manualmente, a continuación emplean una enorme cantidad de tiempo extra en comprender el código creado por sus predecesores, ampliar y refinar el programa. Durante las primeras tres décadas de la era de las computadoras, el reto principal fue el desarrollar hardware que redujera el costo del procesamiento y almacenamiento de los datos. A partir de la década de los sesentas, los avances en microelectrónica han dado como resultado mayor poder de cómputo a un costo cada vez más reducido.

Ahora el problema es diferente. El reto principal es reducir el costo e incrementar la calidad de las soluciones basadas en computadora, que son implantadas con software.

En libros populares acerca de la "Revolución de las Computadoras", Osborne bosquejó una nueva "Revolución Industrial"; Toffler llamó al advenimiento de la microelectrónica como "La tercera ola del cambio en la Humanidad"; y Najshitt predijo que "la transformación de una sociedad industrial a una sociedad de información tendrá un profundo impacto en nuestras vidas". Durante los primeros

años del desarrollo de sistemas de cómputo, el hardware continuó bajo cambios contínuos mientras que el software fue visto por muchos como un objetivo posterior. La programación de computadoras fue vista como un arte, por lo cual la única técnica de programación disponible consistió en definir un algoritmo y tratar de reproducirlo fielmente en el lenguaje de programación disponible para el computador. El software producto estaba en su infancia. La mayoría del software era desarrollado y finalmente usado por la misma persona u organización. La misma persona escribió, ejecutó y si falló, arregló el programa. Debido a que el cambio de trabajo era mínimo, los administradores sabían donde encontrar a la gente cuando surgía un problema. Por este ambiente de software personalizado, el diseño fue un proceso implícito ejecutado por una persona y sin documentación existente. Cuarenta años después de la invención de la subrutina seguimos construyendo sistemas a mano, es decir, una instrucción a la vez hemos desarrollado mejores métodos para el proceso de construcción de software, lo cual no funciona bien en sistemas grandes y complejos. Adicionalmente estos métodos producen software plagado de defectos que es difícil de modificar o mantener.

Hacia 1970 se estimaba que la demanda de software, entre 1975 y 1985 crecería entre el 21 y el 23% anual, y no podría ser atendida por la capacidad de producción, que apenas aumentaría entre el 11.5 y el 17% anual. En la década de 1970, el Departamento de la Defensa de los Estados Unidos (DoD), el mayor usuario de computadoras del mundo, informó que pagó el 48% de los recursos destinados a la contratación de proyectos de software por sistemas que nunca recibió, el 27% por sistemas que fueron entregados pero que nunca utilizó, el 21% por productos que le fueron entregados con errores importantes, de modo que los tuvo que abandonar o reelaborar(14%) o modificar(7%). unicamente el 4% de los sistemas fueron entregados correctamente.

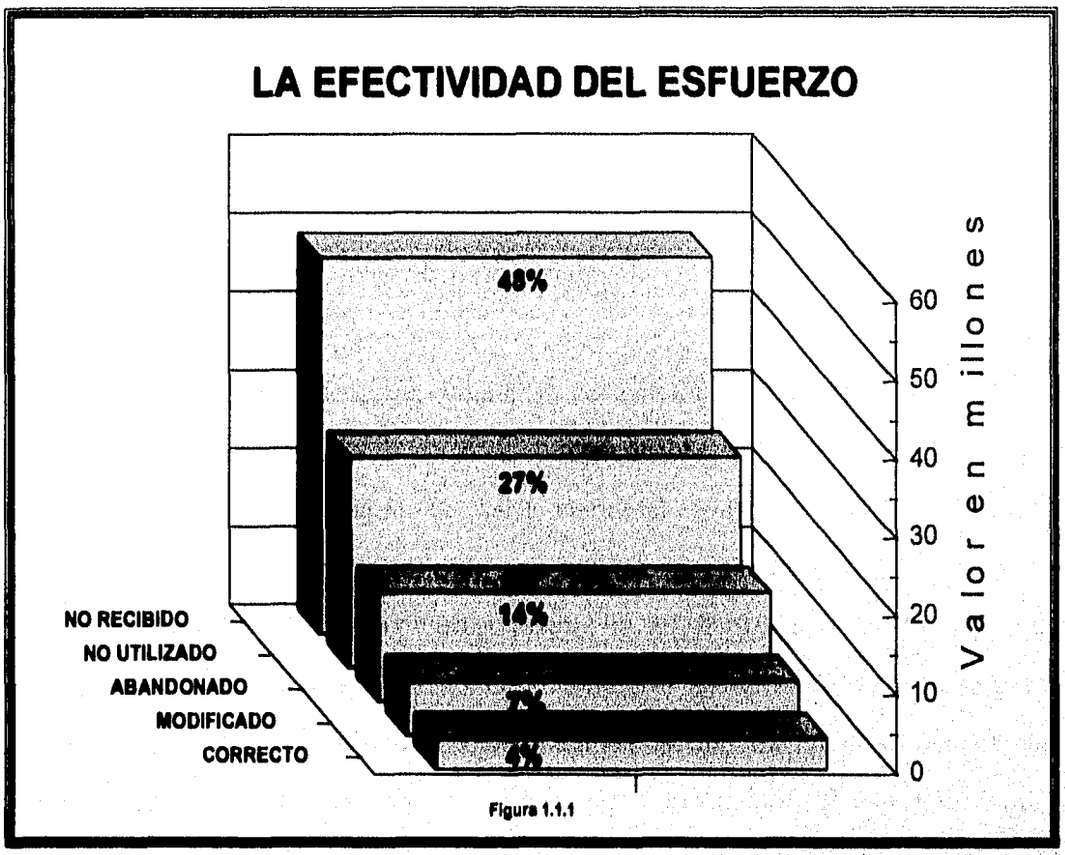


Figura 1.1.1

Por ello Wall Street Journal pudo opinar "El petróleo y el software son los principales obstáculos al progreso económico". Mientras se han logrado algunas mejoras modestas en la calidad y productividad del software, el patrón general de actuación industrial en esta materia sigue siendo el de los calendarios incumplidos, los sobregiros presupuestales y los problemas de calidad. Esta condición de la industria es lo que generalmente se conoce como la crisis del software.

Autores como Feigenbaum y Mc. Corduck predicen que la quinta generación de computadoras y su software tendrán un profundo impacto en el balance político y en el poder industrial en todo el mundo. Ahora, las técnicas de la cuarta generación para desarrollo de software están cambiando el modo en que algunos segmentos de la comunidad del software construyen programas de computadora. Las tecnologías de orientación a objetos, los sistemas expertos y el software de inteligencia artificial finalmente han sido trasladados de los laboratorios a las aplicaciones prácticas para resolver una amplia gama de problemas del mundo real.

La crisis del software continúa aún en nuestros días. Los problemas que caracterizan esta crisis se pueden describir de la siguiente manera:

- La sofisticación del hardware ha sobrepasado nuestra habilidad para construir software que pueda cubrir el potencial del hardware;
- Nuestra habilidad para construir nuevos programas no satisface la demanda de los mismos;
- Nuestra habilidad para mantener los programas existentes es amenazada por un pobre diseño y recursos inadecuados.

Yourdon Metzger y Brooks (empresa líder en implantación de sistemas de computación) han coincidido en señalar que para las aplicaciones de software, se gasta en promedio el 70 % de su presupuesto. Asimismo, señalan que con la finalidad de que el software existente continúe en operación, se requiere darle mantenimiento, a fin de que el tiempo medio para efectuar una modificación o corregir una falla sea menor que el tiempo medio entre dos solicitudes de cambio o entre dos nuevas fallas.

Hoy en día son contados los proyectos de integración que se terminan a tiempo y mucho menos los que lo hacen dentro de lo presupuestado. Pero aún es típico que la mayoría de los sistemas liberados estén plagados de defectos y sean tan rígidamente estructurados, que es casi imposible realizar cambios significativos sin tener que rediseñarlos.

Problemas de costo:

En 1955 el costo del software era menor al 10% del costo del hardware; 20 años después, el costo del software era ya tres veces mayor que el del hardware y para 1985 la diferencia se incrementó nueve veces, es decir exceden los 140 mil millones de dólares. Las siguientes figuras nos representan esta situación:

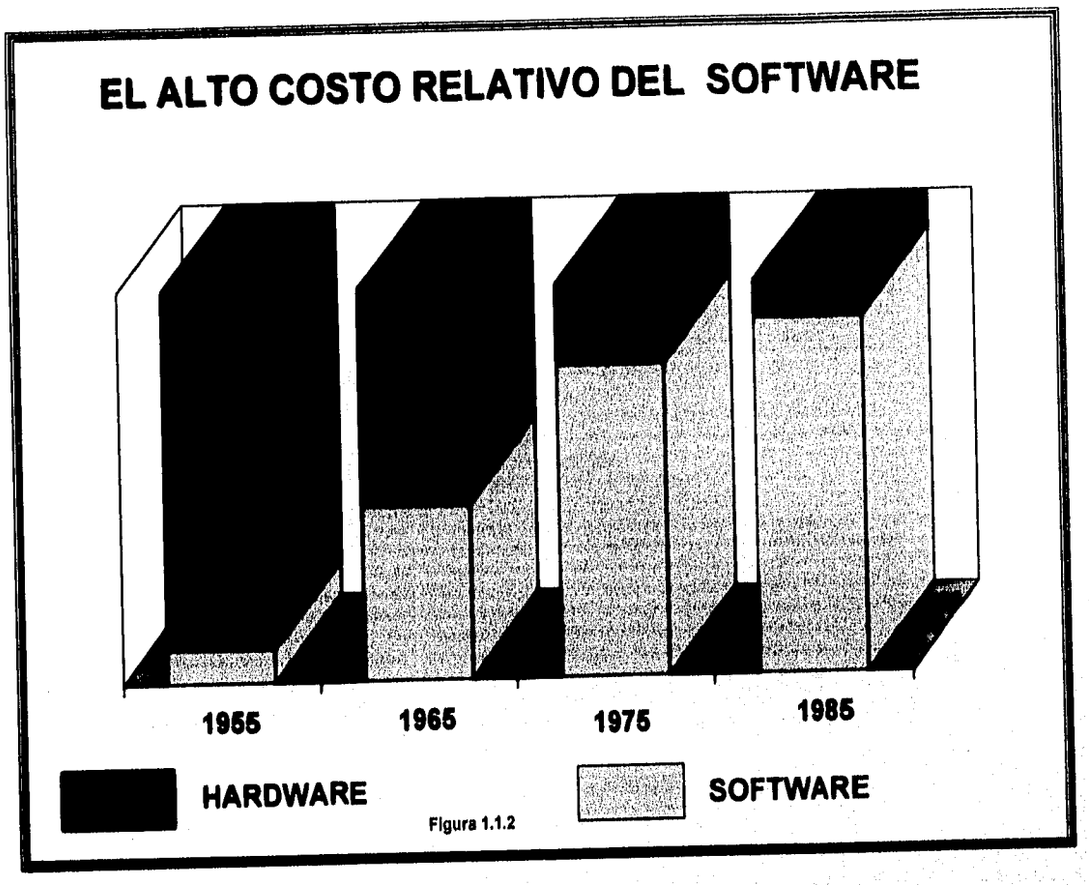


Figura 1.1.2

A partir de entonces han crecido a una tasa de 12% anual, de modo que en 1995 se han alcanzado los 435 mil millones de dólares.

Como refuerzo de lo anterior, el Departamento de Defensa de los Estados Unidos, aseguró que sus costos de software representaban el 45% de su presupuesto informático, frente a un 38% de costos de operación (sobre los que incidía significativamente la mala calidad del software) y sólo un 17% de costos de hardware.

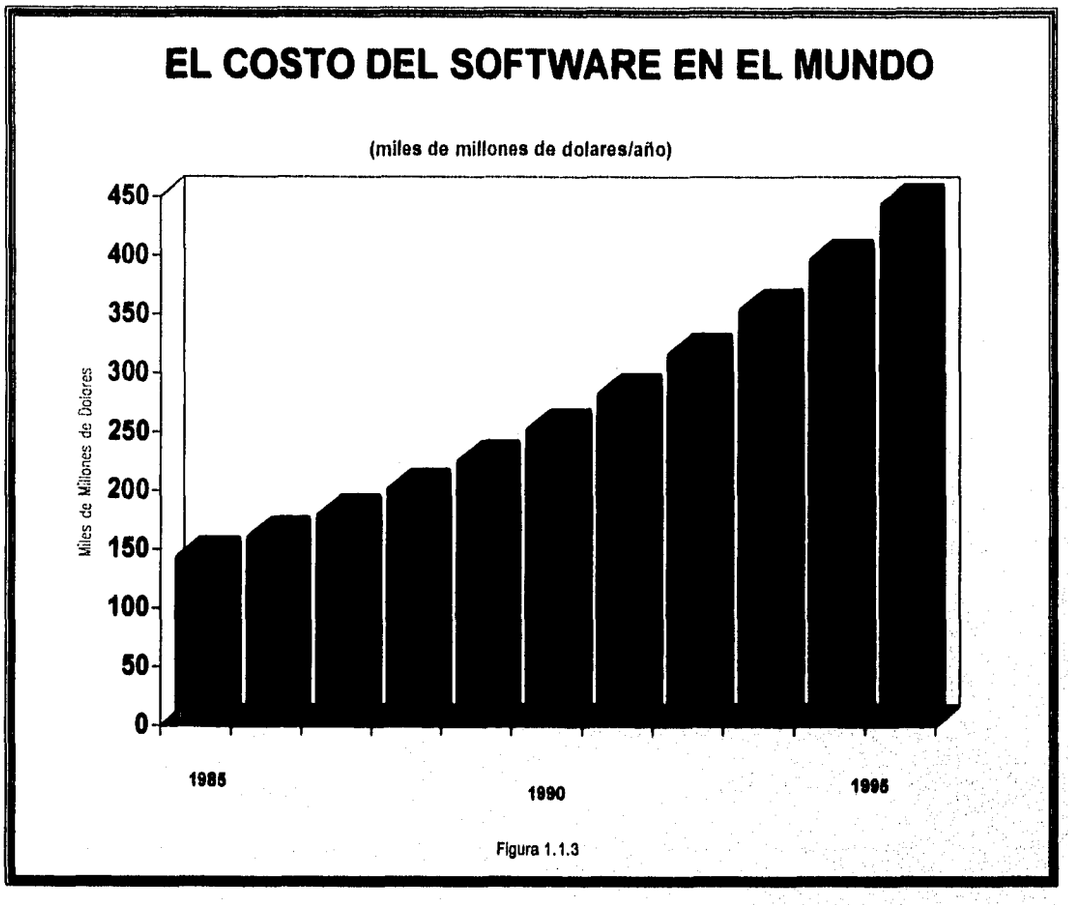


Figura 1.1.3

Según la revista Computer world, "si la industria automovilística hubiera hecho lo que la industria de computadoras ha hecho en los últimos 30 años, un Rolls-Royce costaría dos dólares y medio y rendiría un millón de kilómetros por litro". Se estima que en 1995 el costo del software en el mundo será de 435 mil millones de dólares.

Problemas de confiabilidad:

La crisis del software se caracteriza también por problemas de confiabilidad, veamos a continuación unos ejemplos:

En el verano de 1962, el Mariner I se salió de su curso y tuvo que ser destruído: el problema, que costó entonces 18.5 millones de dólares, se debió a un error en uno de los programas que guiaban la nave. A pesar de que el proyecto Apolo reunió a los mejores programadores de Estados Unidos, un error borró parte de la memoria

de la computadora instalada en el Apolo 8 y se detectaron dieciocho errores de programación en los diez días que duró el vuelo del Apolo 14.

En 1980 un paciente murió a consecuencia de un tratamiento para artritis con una máquina basada en micro ondas, cuyo software reprogramó inadvertidamente su marcapaso; en 1985 y 1986, una falla de software en la máquina de rayos X therac 25 dejó un paciente muerto y varios más paralíticos.

El 15 de enero de 1990, el sistema de larga distancia de la AT&T sufrió una falla catastrófica que dejó paralizada la mayor parte de la red telefónica nacional de los Estados Unidos durante nueve horas. Más tarde se encontró que la falla fue debida al software de encadenamiento.

El diseño deficiente de la interfaz de usuario fue determinado como el factor principal de la identificación incorrecta de una imagen de radar por el USS Vincennes, que resultó en el abatimiento del vuelo Iraní 655 y la muerte de sus 290 pasajeros civiles.

Los anteriores son parte de los grandes problemas que acarrea la falta de calidad en el software.

La solución a dicha crisis es una nueva Ingeniería, "la ingeniería de software", tal como se estableció al fundarse en las conferencias patrocinadas por la OTAN que se celebraron en Garmish (Alemania) en 1968 y en Roma en 1969; su objetivo es superar esta crisis mediante el diseño y la construcción de los sistemas de software como sistemas de ingeniería y no como la programación tradicional que se efectúa en forma meramente artesanal. Los sistemas de ingeniería son sistemas técnicos que aplican el enfoque científico moderno a la formulación y solución de problemas tecnológicos[♦] .

La ingeniería del software con la Orientación a Objetos es el camino que está siguiendo la comunidad informática para tratar de resolver los problemas de la crisis del software.

♦ Para mayor información Favor de consultar : Ingeniería del Software Pressman Mc. Graw Hill

1.2 LOS SISTEMAS DE ARCHIVOS PLANOS

En el principio de la computación las instrucciones de máquina eran muy semejantes, los programas ejecutaban las tareas y nunca las escribían en dispositivos de almacenamiento. En esta etapa uno de los pocos elementos que se almacenaban eran los propios programas, sin embargo, pronto se descubrió el valor de almacenar los resultados, lo cual se incrementó con el advenimiento de almacenamiento en discos rotatorios, lo que ofreció la posibilidad de acceso aleatorio a grandes cantidades de datos almacenados. Con el tiempo, la mayoría de los programas utilizaron este nuevo tipo de almacenamiento en disco, no obstante, los datos en el medio de almacenamiento eran difíciles de organizar y administrar. Esta situación llevó a los diseñadores a crear el sistema operativo, y el software necesario para facilitar el almacenamiento en disco, así nacieron los sistemas de administración de archivos.

Hoy el problema básico de la administración de la información no es su almacenamiento. Virtualmente cualquier clase de información puede almacenarse y recuperarse mediante un archivo de computadora. De hecho el almacenamiento de los datos en los llamados "Archivos planos" a menudo es la mejor forma en que podríamos darle la información al usuario. Los verdaderos problemas empiezan al compartir está información. lamentablemente **los archivos planos no son muy eficientes en el procesamiento multiusuario y con algunos mecanismos de estructuracion de información como lo es el acceso concurrente,** el mantenimiento de la seguridad de información en los archivos planos pueden ser fácilmente leídos, lo que provoca inconsistencia en los resultados o la corrupción de los datos. A principio de los años 60, varias firmas comerciales empezaron a computarizar sus sistemas de información; los datos se guardaban en medios electrónicos en lugar de guardarlos en papel, y se usaban lenguajes de alto nivel para recuperar y manejar los datos.

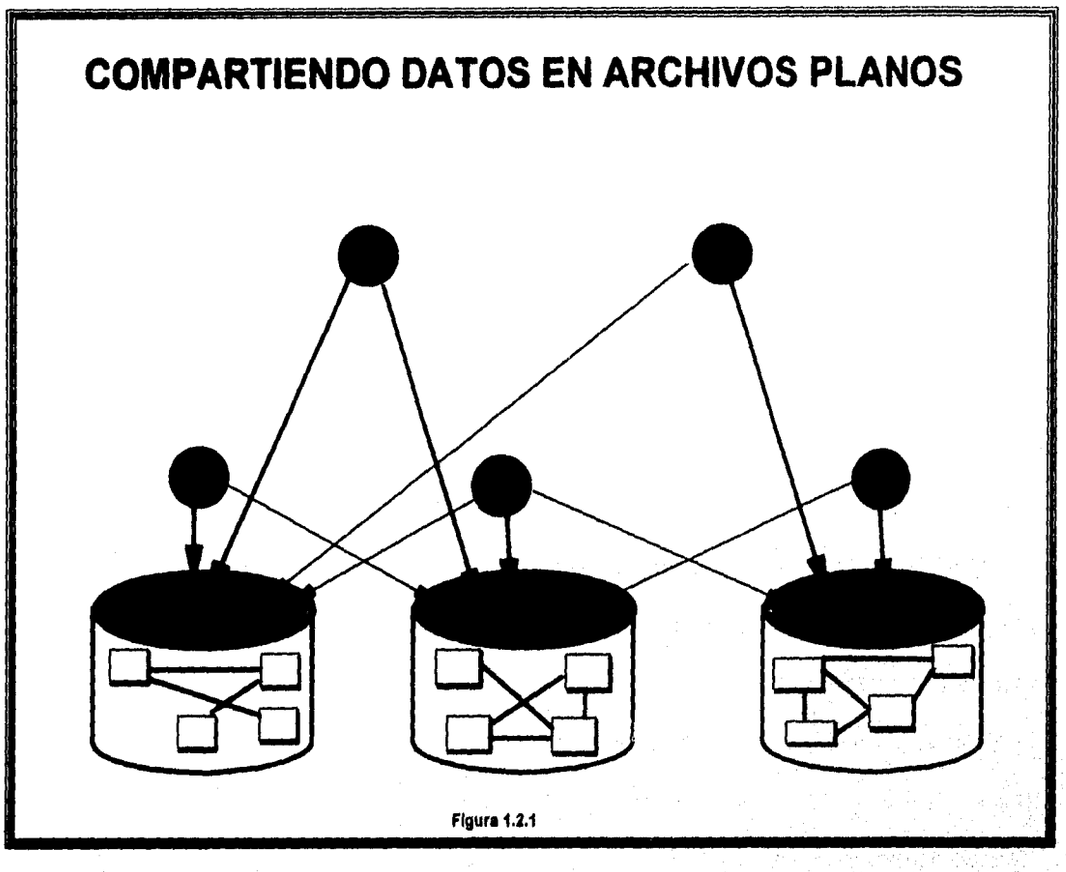


Figura 1.2.1

La figura 1.2.1 muestra como las aplicaciones individuales se desarrollaban en forma independiente y cada programa de aplicación procesaba sus propios archivos privados. Los archivos planos servían solo para una aplicación. Muchas veces los mismos datos podrían servir para otras aplicaciones, pero casi siempre organizados de otro modo lo cual obligaba a la creación de otros archivos. Esto originaba un elevado nivel de redundancia, con varios archivos que en resumen contenían prácticamente los mismos datos.

En un sistema tradicional de procesamiento de archivos planos la recuperación de datos ocurre de la siguiente forma:

1. El programa de aplicación solicita leer un registro de un archivo de datos. El programa de aplicación contiene información sobre la organización del archivo, de la forma en que se accesará, del dispositivo físico donde se encuentran los datos y de la estructura exacta de los campos del archivo.

2. El sistema operativo hace que los datos sean accesados, leídos y transferidos a un buffer en la memoria principal .
3. El sistema operativo transfiere los datos al área de trabajo para el programa de aplicación.
4. Finalmente el programa procesa los datos.

1.3 ALMACENANDO DATOS EN UNA BASE DE DATOS

Cuando la cantidad de datos que se manipulan en un sistema van más allá de los cientos o miles, el hecho de permitir que diversas rutinas los accedan, frecuentemente conduce a errores misteriosos y comportamiento impredecible, dado que siempre existe la posibilidad de que una persona cambie la información que otros están actualizando. En el presente, prevenir esta confusión resulta ser un problema técnico difícil de resolver mediante archivos planos. Al final de los años 60 surgió el sistema de bases de datos para superar los problemas asociados con los sistemas de archivos planos.

Si se habla informalmente de una base de datos, se hace referencia a una colección de datos mutuamente relacionados al hardware de la computadora que se emplea para almacenarla y a los programas utilizados para manipularla; veamos otras definiciones:

- Alice Y. H. Tsai (data base systems: managament and use), "Una base de datos es una colección de archivos interelacionados creados con un DBMS".
- C. J. Date (An Introduction to Data base Systems), "Una base de datos es una colección de datos de operación almacenados y utilizados por los sistemas de aplicación de una empresa específica."
- IBM Data Processing División (Data Processing Glossary), "Conjunto de datos, parte del total de otro conjunto de datos, que es suficiente para un propósito dado o para un sistema de procesamiento de datos dado".

- Ronald Ross (Data Managament Systems), "Una colección de elementos de datos o registros únicos interelacionados, en uno o más archivos de computadora, los cuales pueden ser procesados por múltiples programas de aplicación".
- Leo Cohen (Data Managament Systems), "(a) Los elementos de datos agregados que comprenden el conjunto de archivos y registros de un sistema dado o de un conjunto de sistemas. (b) Un conjunto de archivos que tienen interpelaciones lógicas".
- James Martín (Principles of Data-Base Managament y Computer Data-Base Organization), "Una colección de datos interelacionados, almacenados juntos con una redundancia controlada para servir a una o más aplicaciones; los datos son almacenados de esta manera para que sean independientes de los programas que los usan; se emplea un enfoque común y controlado en la adición de nuevos datos, y en la modificación y recuperación de los datos ya existentes en la base de datos".

La finalidad de una base de datos es la de servir a una o más aplicaciones de la mejor manera posible, el objetivo ideal de una BD es el modelar completamente la estructura de información de una empresa o departamento; además, el propósito de una base de datos es, por definición, la resolución de problemas distintos, algunos de los cuales ni siquiera están presentes en el momento de establecer la base de datos. Los datos representan conocimiento acerca de una empresa, universidad, etc. La organización de los datos en una base de datos, debe representar el significado de fondo o semántica de ellos en forma correcta o eficiente. El desarrollo de una operación substancial de base de datos, invariablemente es una tarea que requiere la cooperación de diversos individuos como son el analista, el usuario, el diseñador de la base de datos, el administrador de la base de datos, etc.

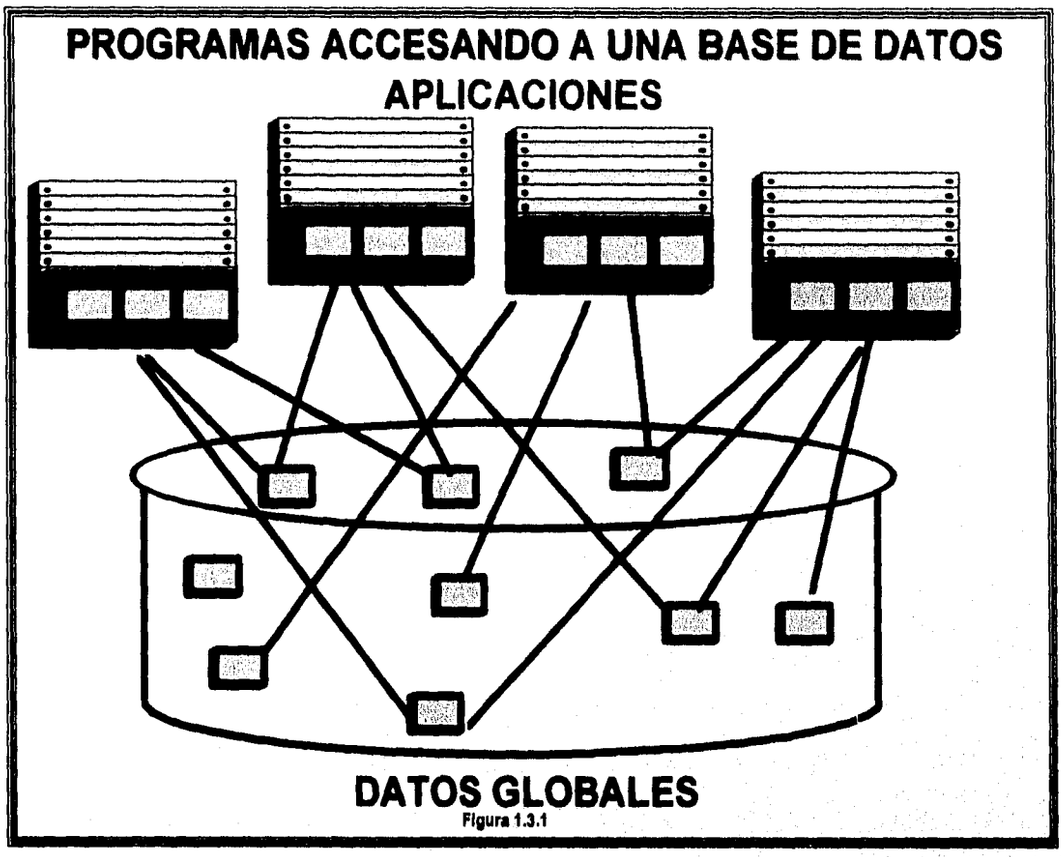


Figura 1.3.1

Las bases de datos relacionales permiten guardar en forma organizada una gran cantidad de información como centro, alrededor del cual giran los programas de aplicación, como se ve en la figura anterior. Es decir, es perfectamente factible que exista información (datos) a pesar de no existir todos los programas que la manipulen. La ventaja más importante (desde el punto de vista relacional) se logra porque todos los programas accesan un conjunto de información común (en vez de que cada programa tenga sus archivos ad hoc, que nadie más usa como en el procesamiento de información con archivos planos). La información se vuelve un recurso compartido . Además de evitar duplicidades, se evitan inconsistencias y se ahorra dinero, ya que la información se guarda en un lugar único de las bases de datos, y sólo se capta una vez .

Sistemas

Un sistema es un conjunto de elementos físicos o abstractos denominados partes que se interelacionan en cierto modo todos entre sí, para lograr un mismo fin u objetivo.

Esto conlleva a la teoría de relatividad de los sistemas, la cual especifica que:

Todo sistema sometido a la influencia de su ambiente es un subsistema de un sistema más grande, y toda parte de un sistema es potencialmente un sistema. Dicho lo anterior la Ingeniería de software nos brinda el enfoque de sistemas, el cual parte del criterio de que para desarrollar un producto de software debe analizarse éste concibiéndolo como sistema, para así definir un conjunto de subsistemas y estudiarlos por separado para luego tratar de definir las condiciones que el sistema impone a los subsistemas, de modo que sea factible el análisis y construcción del sistema mismo.

Los sistemas económicos u organizables son conjuntos de recursos humanos, materiales, energéticos, financieros y de información con relaciones entre sí, con un objetivo puramente económico. De este sistema es posible distinguir un subsistema que es el centro de control y este es el sistema de procesamiento de datos SPD.

Un SPD es entonces parte de una organización, el cual se encarga de almacenar y procesar los datos dentro de la organización.

La forma de este sistema esta dada por el modelo que tenga la organización, así un SPD puede contener a uno o más subsistemas de la organización, como puede ser el sistema de toma de decisiones o el sistema de control de calidad, el sistema de nóminas, etc.

El SPD esta formado por un conjunto de datos (los cuales deben estar agrupados en conjunto) y los procesos de manipulación de éstos. Al medio donde se depositan estos datos se le conoce como base de datos lógica (BDL), la cual se compone de archivos lógicos A.L. y estos de registros lógicos R.L., los cuales se forman de un conjunto de datos.

¿ Que es un Sistema de Base de Datos ?

Un sistema de base de datos es un sistema computarizado de información para el manejo de datos desarrollado y administrado por medio de productos de software, llamados manejadores de bases de datos DBMS. Los componentes principales de un sistema de bases de datos son tres: El hardware, el software DBMS y los datos a manejar.

El sistema de base de datos hace uso de ciertos modelos, metodologías y técnicas que nos ayudan a estructurar los datos y de ciertas herramientas que nos permiten manejar estas estructuras de datos.

1.4 DBMS MANEJADORES DE BASE DE DATOS

En el ambiente de base de datos podemos identificar dos importantes entes, los cuales estando íntimamente relacionados son conceptualmente distintos, ellos son la base de datos y el DBMS (del inglés **D**ata **B**ase **M**anagement **S**ystem) manejador de base de datos.

El concepto base de datos ya se definió con anterioridad, ahora veamos qué es un DBMS. **Un Manejador de Base de datos es un complicado software encargado de proveer a diversos usuarios de una interfaz que les permita comunicarse fácilmente con la base de datos física** (Archivos donde se almacena la información), que corre bajo un sistema operativo, contiene rutinas relacionadas que efectúan funciones especializadas para el procesamiento de la base de datos. En la mayoría de los casos el DBMS depende del sistema operativo para el manejo de los datos. La siguiente figura representa un DBMS.

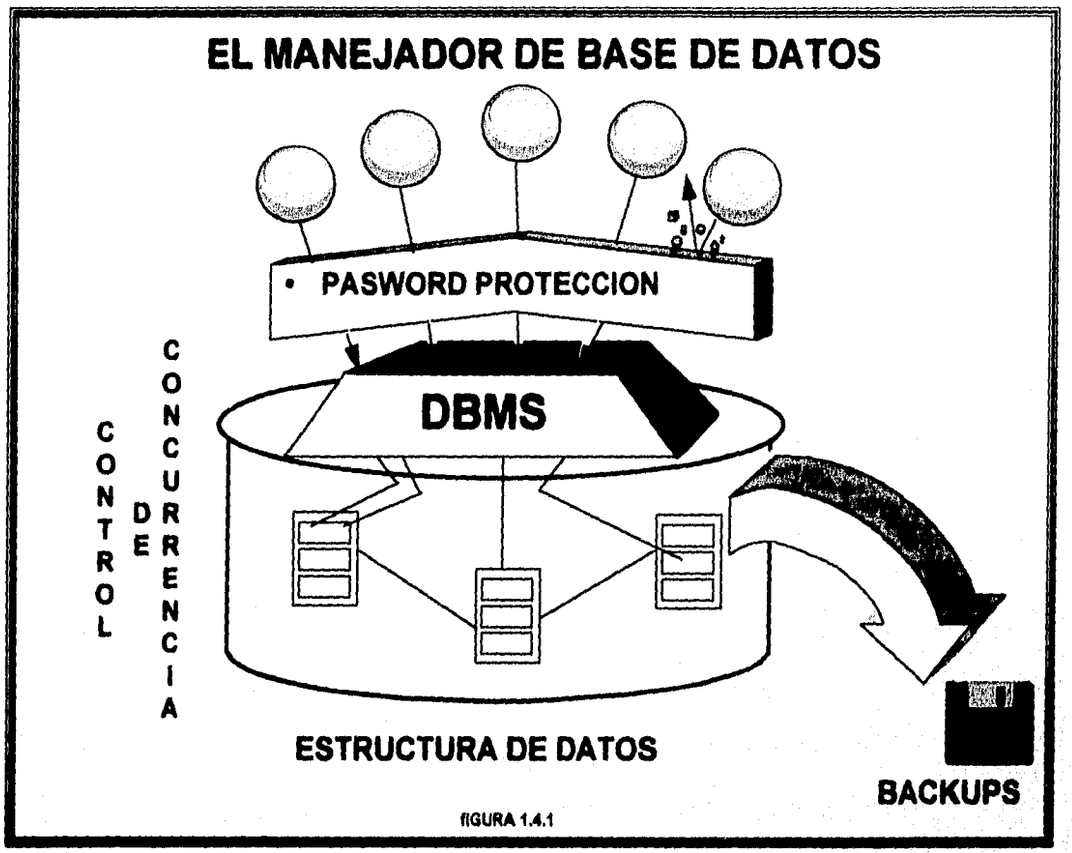


fIGURA 1.4.1

El DBMS le permite al usuario definir, crear, acceder, respaldar, y administrar la base de datos. La tecnología de los DBMS es relativamente nueva, se empezó a utilizar para solucionar los problemas de estructuración de información en archivos planos y regular así el acceso a la información, reemplazando de esta forma los métodos tradicionales de procesamiento de datos con archivos planos, solucionando los problemas mencionados como son:

- **Seguridad.**- Un DBMS requiere paswords para accesar a él, impidiendo así el acceso a la base de datos por usuarios no autorizados, además contiene respaldos automáticos de información, que en caso de corrupción permiten restaurar la información correcta.
- **Integridad.**- La integridad en un DBMS se refiere a que los valores almacenados son correctos y que por lo tanto la base de datos es consistente.

- **Control de concurrencia.-** Un DBMS controla el acceso simultáneo a los datos, que es un aspecto fundamental para preservar la consistencia e integridad de los datos.
- **Estructura.-** Un DBMS provee de mecanismos para que el usuario pueda representar la estructura de los datos, actualizando con facilidad y flexibilidad dicha estructura.

Cabe aclarar que las funciones de seguridad, integridad, y control de concurrencia no se obtienen automáticamente, ya que el DBMS sólo provee las rutinas necesarias para que el usuario haga uso de dichas rutinas para obtener sus beneficios.

El sistema manejador de base de datos DBMS es el componente más importante de un sistema de base de datos, ya que es, el producto de software que permite a los usuarios implementar sistemas sin necesidad de programar mucho. En un sistema de base de datos, la traducción entre la perspectiva global de los datos en la base de datos y la perspectiva local esperada por cada programa de aplicación, es realizada por un software interfaz generalizado conocido como sistema manejador de bases de datos (DBMS). Otra de las funciones principales de un DBMS es permitir a los usuarios trabajar con los datos en términos abstractos, sin necesidad de que ellos entiendan la forma en que la computadora los maneja. En este sentido, el DBMS actúa como un intérprete (de un lenguaje de alto nivel) entre la base de datos y el usuario.

Un DBMS es un producto de software el cual tiene las siguientes funciones:

- Crear y organizar la base de datos.
- Establecer y mantener las trayectorias de acceso a la base de datos, de forma tal que los datos se puedan recuperar rápidamente.
- Manipular los datos de acuerdo a las peticiones de los usuarios.
- Mantener la integridad y seguridad de los datos.

- **Control de concurrencia.-** Un DBMS controla el acceso simultáneo a los datos, que es un aspecto fundamental para preservar la consistencia e integridad de los datos.
- **Estructura.-** Un DBMS provee de mecanismos para que el usuario pueda representar la estructura de los datos, actualizando con facilidad y flexibilidad dicha estructura.

Cabe aclarar que las funciones de seguridad, integridad, y control de concurrencia no se obtienen automáticamente, ya que el DBMS sólo provee las rutinas necesarias para que el usuario haga uso de dichas rutinas para obtener sus beneficios.

El sistema manejador de base de datos DBMS es el componente más importante de un sistema de base de datos, ya que es, el producto de software que permite a los usuarios implementar sistemas sin necesidad de programar mucho. En un sistema de base de datos, la traducción entre la perspectiva global de los datos en la base de datos y la perspectiva local esperada por cada programa de aplicación, es realizada por un software interfaz generalizado conocido como sistema manejador de bases de datos (DBMS). Otra de las funciones principales de un DBMS es permitir a los usuarios trabajar con los datos en términos abstractos, sin necesidad de que ellos entiendan la forma en que la computadora los maneja. En este sentido, el DBMS actúa como un intérprete (de un lenguaje de alto nivel) entre la base de datos y el usuario.

Un DBMS es un producto de software el cual tiene las siguientes funciones:

- Crear y organizar la base de datos.
- Establecer y mantener las trayectorias de acceso a la base de datos, de forma tal que los datos se puedan recuperar rápidamente.
- Manipular los datos de acuerdo a las peticiones de los usuarios.
- Mantener la integridad y seguridad de los datos.

Un DBMS ofrece las siguientes ventajas:

- Los datos se almacenan con una redundancia mínima.
- Se mantiene la integridad de los datos.
- Múltiples usuarios pueden compartir datos de manera eficiente.
- Las aplicaciones son diferentes de como están organizados los datos.
- Se refuerza la seguridad y el uso de los estándares.

ARQUITECTURA DE LAS BASES DE DATOS

La estructura de una base de datos no permanece estática, es decir las empresas van evolucionando, por lo que resulta necesario hacer más eficientes las estructuras actuales; esto es deseable por un lado para que los cambios se realicen de la forma más simple, y por otro que los cambios no se reflejen en el código de los programas de aplicación. Para lograr esto ANSI/X3/SPARC (Standar Planing and Requirements Commitee of the American National Standards Institute on Computers and Information Processing; Comité de Planeación y Requerimientos del Instituto Nacional Estadounidense de Estándares en Computación y Procesamiento de la Información), recomendó que **la arquitectura de una base de datos se visualizara desde tres niveles : Interno, Conceptual y Externo.**

Modelo Interno

El modelo interno es la representación del nivel inferior de una base de datos. Mapea a la base lógica hacia el almacenamiento físico y establece trayectorias de datos mediante índices para el acceso aleatorio a las bases de datos. El modelo interno es descrito por el DBMS como un esquema interno. El esquema contiene especificaciones detalladas del almacenamiento de todos los registros almacenados en la base de datos, así como los descriptores del sistema: índices, palabras de control y trayectorias de datos necesarias para la recuperación sobre claves secundarias. Es la perspectiva del almacenamiento físico.

Asimismo, es la descripción de cómo los datos están almacenados a nivel de registros, formatos, índices, apuntadores, etc.

Modelo Conceptual

El administrador de la base de datos (data base administrator, DBA) define el modelo conceptual . Este modelo representa la visión organizacional de la base de datos que se obtiene al integrar los requerimientos de todos los usuarios en una empresa.
Por ello que **el nivel conceptual es la perspectiva del usuario**, la descripción de cómo los datos de interés para la empresa están almacenados en la base de datos.
La transformación de registros conceptuales en registros físicos se realiza por el DBMS y es totalmente transparente para el usuario.

Modelo Externo

Es la perspectiva de cada programador de la base de datos. Desde el punto de vista del usuario de computadoras, un sistema manejador de la base de datos provee fácil acceso comprendible a un almacén de información. Además, esta información es mantenida por el sistema en un formato definido, desde el cual los reportes impresos son fácilmente obtenidos. Es la descripción de la vista local de la base de datos requerida por una aplicación.

1.5 LA EVOLUCION DE LAS ESTRUCTURAS DE LAS B. D.

De todos los problemas que tienen los DBMS, el más nombrado es el desarrollo de mecanismos para representar con facilidad y flexibilidad la estructura de los datos, ya son más de 40 años de evolución de las bases de datos, principalmente en el desarrollo de nuevas generaciones de tecnologías que representen la estructura de la información.

Se han realizado investigaciones para establecer la mejor manera de almacenar los datos de una aplicación. Estos estudios han dado lugar al nacimiento de un término

de uso generalizado en la base de datos llamado **modelo de datos**. La siguiente figura muestra la clasificación de los modelos más representativos, dicha clasificación se basa en la estructura lógica a nivel conceptual externo.

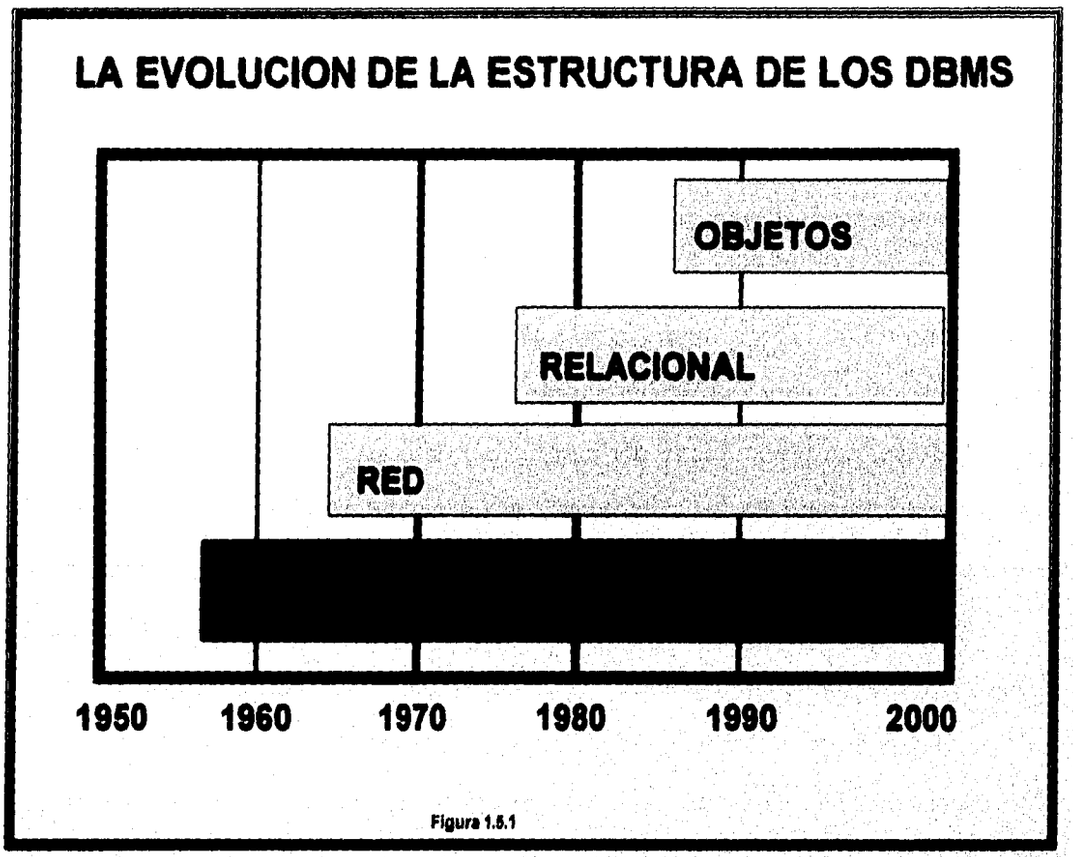


Figura 1.5.1

Son tres modelos de datos comúnmente aceptados: jerárquico, red y relacional. El modelo orientado a objetos esta tomando gran fuerza. Generalmente un DBMS está diseñado para el manejo de uno de esos cuatro tipos de modelos, a continuación veamos los tres primeros modelos con el enfoque que le da el DBMS, el modelo OO lo veremos en los siguientes capítulos.

El principal método para el diseño de una base de datos es la construcción de modelos que representen dicha base de datos en forma tal, que permita la manipulación de los bloques conceptuales de construcción para la base de datos, pero cualquier aplicación tiene que combinarse con sentido común y visión profunda.

El concepto de un DBMS conjunta varias ideas en un sólo sistema los DBMS; se basan en un sólo modelo de datos independiente de cualquier aplicación particular, con lo cual el diseño de los datos se convirtió en la actividad más importante del diseñador, ésto ocasionó un cambio en el paradigma hacia un modelo de desarrollo de aplicaciones orientado hacia los datos.

Modelo Jerárquico

La expresión base de datos comenzó a popularizarse al principio de los sesentas. Antes de esa época, en el mundo de la informática se hablaba de archivos planos y conjuntos de datos. Como ocurre a menudo cuando un nuevo término se pone de moda, no faltaron quienes quisieron promover de categoría sus archivos llamándoles bases de datos, sin preocuparse por cambiar su naturaleza, lo que debió haber sido necesario para dotarlo de las características de no redundancia, independencia de datos, interconectividad, protección de seguridad, y en muchos casos, accesibilidad en tiempo real. Las bases de datos fueron construidas inicialmente para soportar aplicaciones que procesaran grandes volúmenes de datos uniformemente ordenados con una pequeña estructura interna o sin ella.

Al inicio de los años setentas, las bases de datos jerárquicas, se separaron de las aplicaciones y los datos. Los modelos de datos satisfactorios desarrollados para aplicaciones de los años setentas eran similarmente uniformes y simples en estructura. **Los modelos eran con frecuencia orientados a objetos**, con el único elemento de datos en cada campo. Este simple modelo, conocido como modelo jerárquico, permitía el desarrollo de funciones, tales como lenguajes de consulta (Query Lenguajes) ad-doc, optimización de consultas, comprobación de limitaciones y control sobre la gestión de almacenamiento subyacente.

A diferencia del enfoque de red cuyas guías están impuestas por el comité de CODASYL (Conferencia Sobre Lenguajes de Sistemas de Datos), el enfoque jerárquico no tiene normas en la industria. Por ejemplo los sistemas: Information

Managament System (IMS) de IBM y System 2000 de Intel son sistemas jerárquicos, pero difieren enormemente en la recuperación de datos.

Un modelo de datos jerárquico es una estructura familiar. El departamento de personal de una organización exhibe las posiciones de autoridad mediante el empleo de un organigrama jerárquico; los genealogistas utilizan un esquema denominado árbol genealógico para mostrar la descendencia ancestral de una persona, familia o grupo. Los programadores emplean diagramas jerárquicos para representar la estructura de control dentro de un programa .

El modelo de datos jerárquico representa los datos como un árbol invertido de entidades interconectadas. El nodo superior de este modelo se denomina raíz. Otros nodos se ramifican desde la raíz. Ninguna ocurrencia de un nodo puede contener más de un nodo en su nivel superior.

Una analogía existente para poder explicar el modelo jerárquico es la de un árbol genealógico en el cual los padres pueden no tener hijos, tener uno o más de uno. si hay hijos estos mismos pueden tener sus propios hijos. El árbol genealógico es un árbol jerárquico si ignoramos la existencia de un padre en cada nodo. No es accidental que el modelo jerárquico tome gran parte de su terminología de esta analogía. El modelo jerárquico es un modelo que organiza los datos en una estructura jerárquica de árbol la cual se construye con nodos y ramas. Un nodo es una colección de atributos de datos que describen a la entidad en ese nodo. El nodo más alto se conoce como raíz. Cada vez que aparece el nodo raíz da principio un registro lógico. Una base de datos jerárquica esta constituida por varios árboles.

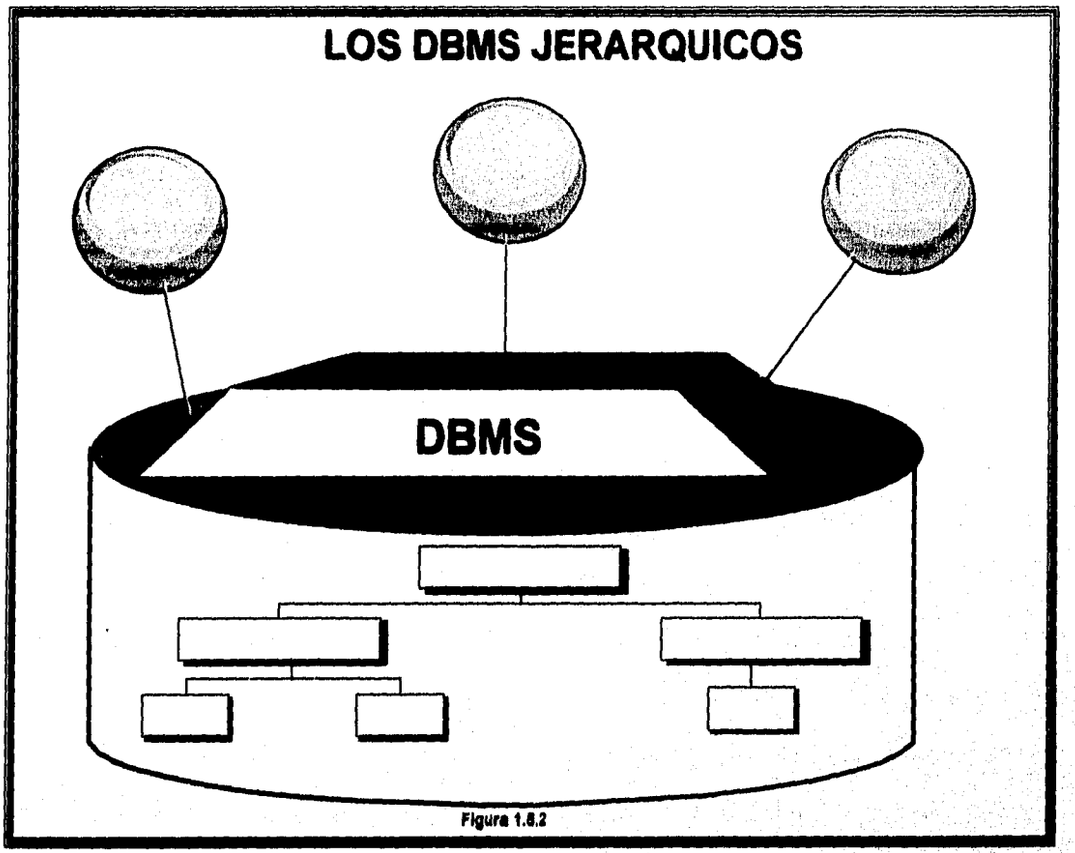


Figura 1.8.2

Un DBMS con enfoque jerárquico maneja solo tres tipos de estructuras. Cada nodo en una estructura de árbol representa un registro de tipo conceptual o un segmento de datos. Una de las restricciones de las estructuras de árbol es que cada nodo no puede tener más de un padre.

El modelo jerárquico es insuficiente para capturar todas las posibles estructuras de información, además de ser muy rígido, en lo que se refiere a cambios en la estructura. Los DBMS más conocidos de este tipo, como se cita anteriormente, son IMS de IBM, y SYSTEM 2000 de Intel, estos dos difieren enormemente; el punto a favor de los sistemas jerárquicos es que fueron los primeros que aparecieron totalmente desarrollados en el mercado. IMS desde que salió al mercado a finales de 1960 para manejar una gran cantidad de datos para investigación aereoespacial, se ha convertido en uno de los DBMS más utilizados. No hay duda que el diseño de IMS es anticuado, no obstante ha sufrido una serie de actualizaciones hasta nuestros días. Con el intento de resistir la competencia, han salido al mercado

versiones revisadas para estar al día con la evolución de los sistemas operativos de IBM tales como: MFT, MVT, VS1, VS2, y MVS. IBM también proporciona auxilios para convertir bases de datos IMS en bases de datos SQL/DS o DB2 para que se les pueda procesar mediante el SQL. Sin embargo, los diseñadores descubrieron que el mundo no era inherentemente jerárquico.

Modelo de Red

Al trabajo de CODASYL y de sus comités, podemos decir que se deben los conceptos de base de datos con estructuras de RED. CODASYL se estableció en 1959 en Washington, D. C., en un encuentro de representantes de 40 importantes usuarios de computadoras, fabricantes y departamentos de gobierno. La intención del comité es desarrollar y recomendar técnicas y lenguajes para el análisis, implantación y operación de sistemas de procesamiento de datos, así como de proporcionar estas especificaciones a los grupos normativos en forma de un informe técnico de desarrollo. Su primera gran tarea fue el desarrollo del lenguaje de programación COBOL . Para que el COBOL pudiera extenderse y pudiera manejar la administración de las bases de datos, el comité CODASYL instituyó un grupo nuevo llamado Grupos de Tareas de Bases de Datos (Data Base Task Group DBTG). El famoso reporte CODASYL DBTG se publicó primero en 1969 y se revisó en 1971.

En una estructura de red, un nodo hijo pude tener mas de un nodo padre. Las relaciones entre entidades se establecen como conjuntos que consisten en un propietario y sus miembros, unidos por apuntadores.

Los DBMS más conocidos son: IDMS de Cullinet, DBMS VAX-11 de DEC, y otros sistemas de peticiones CODASYL. IMAGE de Hewlett Packard es un sistema no CODASYL que puede manejar estructuras simples de red.

Una de las ventajas de IMS sobre los sistemas relacionales es su excepcional rendimiento en bases de datos grandes, especialmente cuando el volumen de transacciones es mayor; sin embargo, el lado negativo es la inflexibilidad para

proporcionar al usuario diferentes vistas o escenarios distintos para consultas con propósitos determinados.

Modelo Relacional

Cuando la administración de bases de datos se popularizó a partir de los sesenta, surgieron varios modelos de datos populares con sus ventajas y desventajas.

El modelo relacional fue definido en 1970 por el Dr. Codd por medio de una serie de reglas cuyo objetivo fue lograr la independencia de la representación lógica de los datos de su almacenamiento físico. Esta independencia física/lógica se refiere a tres aspectos:

1. Independencia de la ordenación.
2. Independencia de la indexación.
3. Independencia de los caminos de acceso.

Por ello, Codd a través de sus reglas pretende los siguiente objetivos:

- Independencia Física Lógica.
- Eliminación de Redundancia.
- Flexibilidad.
- Uniformidad.
- Sencillez.
- Sólido fundamento Teórico.

Actualmente el modelo más popular es el relacional, se estima que el 80% de los DBMS se derivan del modelo relacional, desplazando a los otros modelos tradicionales, principalmente al de red y jerárquico. El modelo relacional ha evolucionado los últimos 25 años permitiendo cada vez el soporte de estructuras cada vez mas complejos. EL modelo relacional consistente en tablas es simple y comprensible. Los métodos para manipular los datos en las tablas son bien conocidos y razonablemente eficaces. El álgebra relacional proporciona una base

matemática formal capaz de soportar las metodología de análisis de datos. Por último, con la posibilidad de unir dinámicamente los datos en tablas virtuales durante la ejecución, las bases de datos relacionales facilitaron el desarrollo de aplicaciones con un nivel más alto de independencia en los datos, que los modelos predecesores.

Una base de datos relacional se organiza en tablas llamadas relaciones, cada una de las cuales se implanta o representa físicamente como un archivo. No existe necesariamente una relación lineal (uno a uno) entre las tablas y su representación física en términos de archivos. De hecho, una tabla puede corresponder a más de un archivo y varias tablas pueden estar almacenadas físicamente en un mismo archivo. En la jerga relacional una fila (renglón) en una relación se llama tupla o ada y representa un registro o una entidad; cada columna en una relación representa un campo o un atributo. Así una relación se compone de una colección de entidades (o registros) los cuales están descritos por cierto número de atributos predeterminados implantados como campos. Se denomina grado de una relación al número de atributos, y al número de tuplas de una relación se le llama cardinalidad. Una clave candidata es aquella que identifica de manera inequívoca un registro. Evidentemente pueden existir varias combinaciones de atributos que satisfagan el criterio anterior. Una de ellas se selecciona y se asigna como clave primaria. Al resto de las claves se les llama claves alternas.

A un atributo se le llama clave externa si tiene como restricción que los valores que puede tomar existen en otra relación como clave primaria. Si en un registro hay un atributo sin ningún valor, entonces se dice que dicho atributo tiene valor nulo. Un campo con valor nulo es totalmente diferente a un campo con todo a ceros o blancos. Un modelo relacional de bases de datos se compone de relaciones, y las relaciones entre ellas se establecen implícitamente vía claves externas, es decir, la relación entre dos archivos en el sistema relacional se establece implícitamente por la presencia de un campo común en ambos archivos.

Las bases de datos relacionales son más flexibles que sus predecesoras, ya que diversos programas diferentes extraen diferentes niveles de altas estructuras. Esto también es una nueva clase de información que es agregada a la base de datos.

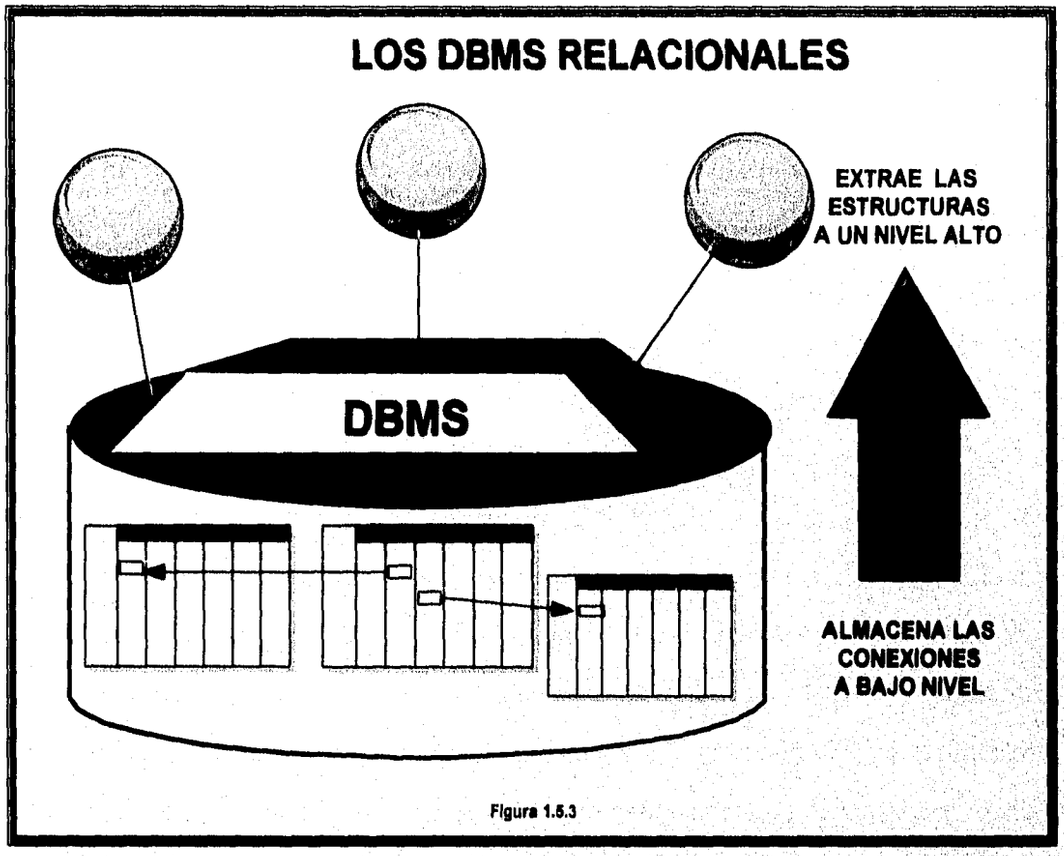


Figura 1.5.3

Pero esta tecnología tiende a ser ineficiente al manejar estructuras complejas que al mismo tiempo los sistemas requieren en los procesos de construcción de sistemas de información.

La información almacenada en computadoras es usualmente dividida en grupos, (que comúnmente se les denomina archivos -files), el poder obtener información combinada (es decir de más de un archivo), para un reporte en particular, se le llama base de datos relacional, para ello es necesario que los grupos tengan un elemento en común.

Un **D**ata **B**ase **M**anagement **S**ystem Relacional típico incluye:

1) EL DDL. - LENGUAJE DE DEFINICION DE DATOS-

El DBMS permite que el usuario pueda integrar una base de datos de acuerdo a los requerimientos específicos del sistema, el Administrador de la Base de Datos debe definir la estructura de la base de datos en el nivel externo, usando el DDL proporcionado por el DBMS.

2) El DML. - LENGUAJE DE MANIPULACION DE DATOS-

El DBMS proporciona un lenguaje de manipulación de datos para que genere comandos de entrada/salida para accesar a la base de datos. El DML puede ser una extensión del lenguaje de computadora nativo o central o un lenguaje totalmente independiente. Las sentencias del lenguaje DML de algunos sistemas relacionales se pueden tipear desde una terminal en forma interactiva o se pueden incluir en un programa generado con un lenguaje de alto nivel como si fuera parte del mismo lenguaje.

La mayoría de los lenguajes DML CODASYL son esencialmente, una extensión de COBOL. Los DML para el modelo relacional, jerárquico y orientado a objetos difieren en muchas formas.

3) SQL. -LENGUAJE DE CONSULTA ESTRUCTURADO-

En la década de los setentas se desarrollaron tres prototipos importantes basados en el modelo relacional: SYSTEM R, INGRES y QUERY BY EXAMPLE. Uno de ellos SYSTEM R utilizaba para el acceso a la base de datos un lenguaje llamado Structured Query Languaje, conocido también como SIQUEL el cual evolucionó a SQL. Como resultado de una investigación de bases de datos en IBM Research Center en San José, California SQL fue incorporado con el tiempo a productos con un modelo relacional como SQL/DS y DB2. A partir de ese momento todas las versiones de DBMSR obedecen exactamente al SQL estandar de ANSI; sin embargo, múltiples fabricantes han definido algunas extensiones al lenguaje que lo hacen más poderoso.

La vista o subesquema es una porción del modelo abstracto. El término esquemas es usado para referirse a los planos o estructuras de la B.D. Instancias, es el contenido actual de la B.D.

Desafortunadamente se puede demostrar que el SQL no es lo suficiente poderoso para expresar una consulta que obtenga todas las subpartes de una bicicleta a través de una cerradura transitiva. Lo más que se puede hacer es obtener un árbol con una profundidad máxima fija.

Select ...

from Composicion var 1,..., Composicion var n

where var1. subparte ID = var 2.Superparte ID

...

and var n-1 Subparte ID = var nSuperparte ID.

La formulacion de una consulta que produzca un árbol con una profundidad arbitraria es imposible en SQL. Para efectuar esta operación, se debe utilizar un lenguaje de tercera o cuarta generación que tenga SQL embebido.

DESVENTAJAS DEL USO DE LLAVES COMO IDENTIFICADORES

Modificación de llaves: Las llaves primarias como identificadores no pueden ser modificadas, aún cuando son atributos descritos o definidos por los usuarios. Es común utilizar atributos que almacenan nombres como llaves o identificadores, si el valor de estos atributos cambia por alguna razón, por ejemplo al ser corregidos o abreviados, todas aquellas tuplas que hacían referencia a esa tupla se verán afectadas.

Poca uniformidad: Los principales motivos por no tener uniformidad:

El enorme esfuerzo invertido en la optimización de los sistemas relacionales, muy cercano a los modelos de red, sugiere que los proveedores de éstos, incrementarán su porcentaje en el mercado. Los principales DBMSR estan ampliando el modelo para soportar objetos, pero la filosofía sigue siendo la misma: el modelar los datos relacionalmente. Sin embargo, se espera que los DBMS Orientados a Objetos crezcan en el mercado de tecnología de información.

Limitaciones de los DBMS Relacionales

Sin embargo, el modelo relacional sufre al menos de un inconveniente notable. Es difícil expresar la semántica de objetos complejos con sólo un modelo de tabla para el almacenamiento de datos. Aunque las bases de datos relacionales son adecuadas para contabilidad u otras aplicaciones típicas de procesamiento de transacciones en las que los tipos de datos son simples y escasos en número, el modelo relacional ofrece ayuda limitada cuando los tipos se vuelven numerosos y complejos.

El modelo relacional no es una panacea,*[1] las principales áreas en las que presenta deficiencias "al menos en su formulación original" son:

1. Consultas recursivas.
2. La información faltante (si cambiamos el valor de una llave).
3. Conocimientos "semánticos".
   El modelo relacional es plano (Semántica pobre).
4. Manejo de Objetos complejos, dominios fijos, ausencia de extensibilidad (tipos de datos definidos por el usuario).
5 Impedimento "mismatch" entre el DML y Lenguaje de Programación.- Son dos lenguajes con diferentes filosofías, "declarativo" vs "procedual", orientado a "conjuntos" vs un "registro a la vez", conversiones entre dos mundos diferentes (variables).
6 Datos sin operaciones.

[1] Para mayor información ver C.J. Date Introduccion a los sistemas de bases de Datos, Edit. Adison Wesley Iberoamericana. Capitulo 15, 23 y 24.

7 Funcionalidades de los DBMSR cerradas.

Los requerimientos de las aplicaciones tradicionales y no tradicionales NO son totalmente satisfechos por un DBMS Relacional. Sin embargo, el SQL es un standard.

La siguiente figura nos muestra el dilema de las bases de datos:

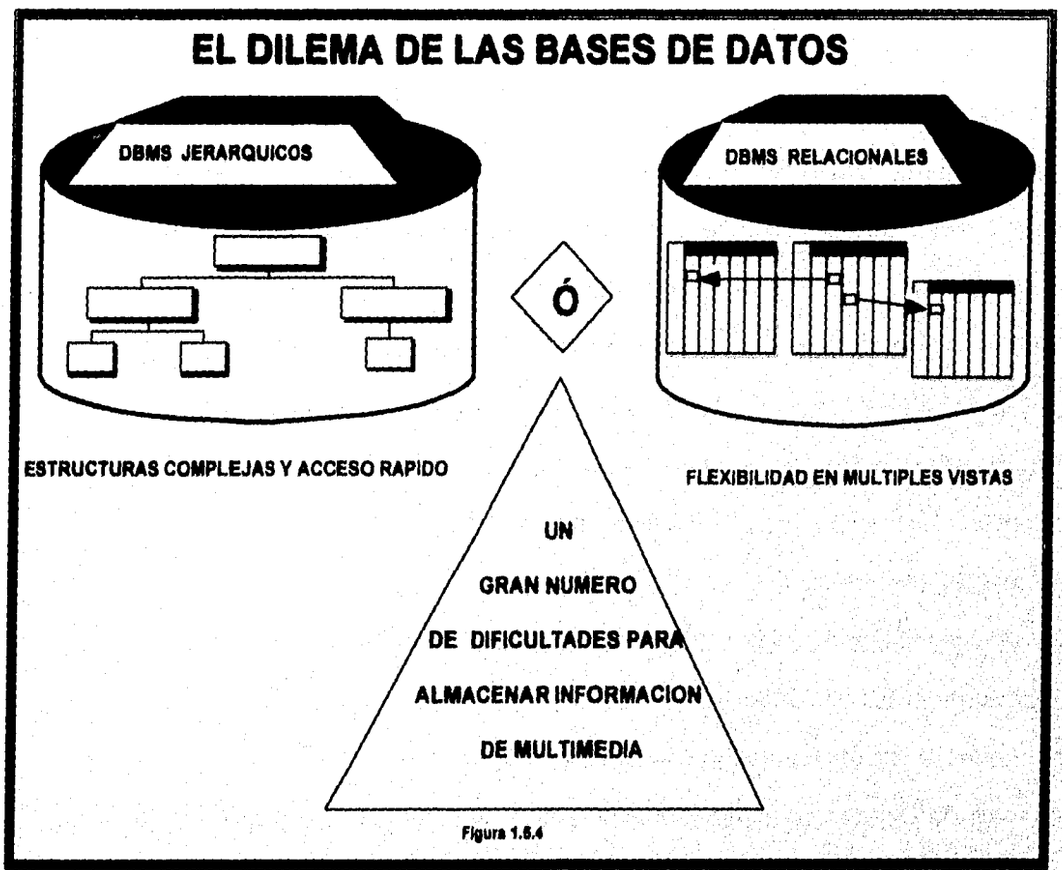


Figura 1.5.4

A pesar de que el modelo relacional es más flexible que sus predecesores, paga un precio por está flexibilidad: la información sobre relaciones complejas tiene que ser expresada como procedimientos en cada programa que acceda la base de datos, y a mayor complejidad corresponde mayor penalización en los tiempos de acceso, ya que las estructuras de datos deseadas tienen que ensamblarse cada vez que se acceden los datos.

Una base de datos de red o jerárquica puede almacenar objetos complejos, pero esta arquitectura no es flexible, lo cual motiva al uso del modelo relacional, el problema principal con los modelos de red o jerárquicos es que la estructura es definida rígidamente cuando la base de datos es creada, por lo que estos sistemas no permiten flexibilidad para modificaciones; por otro lado, una base de datos relacional tiene una estructura más flexible, pero no puede manejar tipos de datos complejos.
Obviamente existen varias formas de solucionar estos problemas causados por tales deficiencias:

1. **Ampliar el modelo relacional** en forma apropiada (solución evolutiva).
2. Desechar el modelo relacional por completo y **sustituirlo por algo nuevo** (el orientado a objetos solución revolucionaria).

Analicemos también los costos y riesgos del modelo relacional:

En el modelo relacional se requiere que el dominio de los atributos contenga solamente valores simples en el sentido de que esos valores no pueden tener estructura interna (deben ser valores atómicos), es posible trabajar en dominios de listas, conjuntos, arreglos, etc. Un renglón en una tabla tiene sólo valores atómicos, se tiene una estructura muy simple, y se obtiene una implementación muy eficiente. Si al menos uno de los atributos tiene un dominio con estructura como listas, tuplas, etc., se debe hacer un proceso de normalización, el cual separa los atributos de un objeto en más de una tabla.

La tecnología orientada a objetos es sólo uno de los varios intentos de hacer frente a los problemas mencionados en este capítulo sobre las defectos del modelo relacional. Otros de ellos son: el modelo semántico, los sistemas de bases de conocimientos, las relaciones anidadas, los sistemas extensibles y los sistemas basados en diversos modelos funcionales. Ahora bien, existen ciertas áreas de traslapo entre estos diferentes enfoques y no siempre es fácil decir donde termina uno y donde comienza el siguiente; sin embargo, es importante intentar caracterizar las principales diferencias:

1. **Modelo Semántico.**[*] - La idea fundamental de este modelo es tratar de identificar un conjunto de construcciones que parezcan útiles de manera genérica, en el sentido de que se presenten en alguna forma o variedad en una amplia gama de aplicaciones. Como ejemplos de estas construcciones podemos mencionar: claves primarias, claves ajenas y dominios, (a nivel del modelo relacional básico), o los núcleos, características, y asociaciones, etc. Si podemos formalizar e identificar tales construcciones y operadores para esas construcciones, podremos integrarlas en nuevos sistemas de bases de datos y así hacerlos más inteligentes, sin embargo, el modelo semántico en este sentido parece ser muy difícil, y los intentos de ofrecer construcciones de utilidad genérica en un nivel más alto de abstracción que el del modelo relacional básico sólo ha tenido un éxito limitado hasta la fecha (con la excepción, claro, de que algunos de esos intentos han tenido un impacto considerable sobre las metodologías de diseño de base de datos). Ahora bien, la tecnología OO trata de lograr que los sistemas de base de datos se muestren más inteligentes. Pero la inteligencia hasta donde puede hablarse de ella, no reside en construcciones entendidas en forma genérica, por ejemplo las características al estilo del Modelo relacional extendido, sino mas bien en objetos encapsulados, que deben haber sido definidos por algún especialista técnico, es decir, si un objeto departamento sabe lo que significa su método, "añadir un empleado" quiere decir en realidad que algún especialista ha escrito un código para ejecutar esa operación, y ha hecho ese código parte de la definición del objeto departamento. El DBMSOO no sabe en sí lo que significa añadir un empleado. (en particular el optimizador no lo sabe).

2. **Sistemas de bases de conocimiento.**- Este es un término sin un significado muy preciso, o al menos sin un significado de aceptación universal. En ocasiones se

[*] Para mayor profundización sobre el tema consultar C.J. Date. Introducción a los sistemas de bases de datos. Edit. Adison Wesley.

utiliza para referirse a sistemas que manejan "objetos" en el sentido OO. No obstante, aun cuando abarque objetos el término siempre implica también un manejo de reglas lógicas de inferencia (sistemas expertos).

3. **Relaciones anidadas.**- Varios investigadores han sugerido que una forma de aumentar la utilidad y funcionalidad del modelo relacional es desechar el requisito de que las relaciones estén normalizadas (es decir que estén por lo menos en primera forma normal* Esta idea se conoce con diferentes nombres, entre ellos "relaciones anidadas", "relaciones no normalizadas" o "relaciones NF cuadrada" proviene de las iniciales en inglés de "no en primera forma normal" -non first normal form, NFNF-). Se ha propuesto un álgebra y un cálculo, e incluso (paradójicamente) una forma normal para relaciones NF cuadrada. El objetivo de esta NF cuadrada es el de subsanar algunas deficiencias del modelo relacional por lo tanto, tiene el mismo objetivo que el enfoque OO.

4. **Sistemas extensibles.**- En los laboratorios de investigación se están desarrollando varios proyectos sobre sistemas extensibles de diversos tipos. La idea básica en todos los casos, es que sea posible ofrecer a los usuarios características integradas capaces de apoyarlos en lo que se requiera en cualquier momento, por lo tanto, es necesario ofrecer a los usuarios la posibilidad de adaptar el sistema de diversas maneras mediante la definición de sus propios tipos de datos, funciones, métodos de acceso, estructuras de almacenamiento, etc. Y por supuesto, el objetivo del enfoque OO es lograr precisamente que el usuario pueda construir sus propios sistemas a la medida de sus necesidades, lo cual podrá ser posible dentro de un marco de referencia relacional, o bien desechando por completo este marco de referencia relacional. De hecho el enfoque OO lo desecha en su totalidad.

---

* Para mayor informacion sobre el proceso de normalización ver Introducción a los Sistemas de Bases de Datos E. J. Date Edit. Adison Wesley Iberoamericana.

5. **Modelos funcionales.**- Se han presentado varios intentos para construir sistemas basados en funciones, en lugar de sistemas basados en relaciones, y una vez más el impulso de tales intentos ha sido subsanar algunas de las deficiencias del modelo relacional original, en el enfoque OO está contenido el modelo funcional.

1.6 OTROS CONCEPTOS REFERENTES A LAS BASES DE DATOS

A continuación se analizan algunos conceptos inherentes a las bases de datos.

¿ Qué es una base de datos con conocimiento ?

Una base de datos con conocimiento es parte de la 5ª generación de sistemas computacionales que encabeza la inteligencia artificial, es otra forma de base de datos activa, que poseen conocimiento, los que provienen de la inteligencia artificial (Especificamente de los Sistemas Expertos). Se deseaba que se almacenara conocimiento y a éste se le consideraba activo, mientras que los datos o información eran pasivos. Se pensaba que el conocimiento debería estar formado por hechos y reglas que pudiesen implementarse en una computadora. Mediante una máquina de inferencia se identificaron y concatenaron las reglas adecuadas para razonar un problema, la máquina de inferencia utilizaba la concatenación hacia atrás o hacia adelante (o bien ambas). Este poderoso concepto se aplicó con éxito a ciertos problemas altamente complejos. Este tipo de cómputo necesitaba una base de conocimiento que almacenase tanto datos como reglas y utilizara una máquina de inferencia para buscar reglas y efectuar el razonamiento automático. Los sistemas con base de conocimiento se construyeron en las compañías de inteligencia artificial.

¿ Qué es una base de datos distribuida ?

Las bases de datos distribuidas son aquellas bases de datos que guardan su información en dispositivos de memoria permanente (discos por lo común) que están localizados o asignados a diversa computadoras. Uno de los problemas principales es el de la consistencia de la información, ya que al estar atendiéndose a mensajes provenientes de clientes remotos, puede ser que lleguen fuera de secuencia, y que el mensaje que se generó a las 13:15 hrs. digamos, llegue antes de uno que se generó a las 13:14 hrs. pero que viene de muy lejos.

¿ Qué es una base de datos de tiempo real ?

Puede parecer que las bases de datos distribuidas constituyan el método preferido en los sistemas de tiempo real, debido a que la multitarea es muy común y que los datos se procesan frecuentemente en paralelo, si la base de datos es distribuida y no centralizada, las tareas individuales pueden acceder a sus datos de forma más rápida, fiable y con menos cuellos de botella. El uso de una base de datos distribuida para aplicaciones en tiempo real divide el "tráfico" de entrada/salida y acorta las colas de las tareas en espera, para acceder a una base de datos. Además, la falla de una base de datos raramente causará la falla del sistema entero si se construyen con redundancia. Las eficiencias del rendimiento conseguido mediante el uso de una base de datos distribuida, debe ser analizada frente a cualquier problema potencial asociado con la partición y replicación de los datos. Aunque la redundancia de los datos mejorará el tiempo de respuesta, suministrando múltiples fuentes de información, los requisitos de replicación para los archivos distribuidos también producen problemas logísticos y de sobrecarga, puesto que todas las copias de los archivos deben ser actualizadas. Además del uso de bases de datos distribuidas introduce el problema de control de concurrencia, lo que implica la sincronización de las bases de datos de forma que todas las copias tengan la misma y correcta información disponible para los accesos .

Servidores y Clientes

Con el advenimiento de las redes de cómputo, se ha generado el concepto de un servidor. Este es un programa (residente en una máquina de la red) que proporciona un servicio determinado (piénsese en el programa que imprime archivos a través de la impresora). Este programa sirve a varios usuarios (clientes) a la vez.

Un cliente no carga o se liga a otro cliente; en vez de esto, el programa "cliente" envía un mensaje al servidor, donde se le indica qué tipo de servicio solicita, y donde se le proporcionan los argumentos (nombre del archivo, etc.) necesarios para poder efectuar ese servicio. Una de las labores iniciales del servidor es verificar la identificación del cliente, para ver si tiene acceso al servicio. Si el servidor es un servidor de licencias de un editor de texto moderno, éste tiene la tarea adicional de contar cuántas licencias ya se han usado (a cuantos clientes o CPU's distintas se esta sirviendo), a fin de negar el servicio si el usuario no ha comprado el número de licencias necesarias.

La máquina de base de datos

Una máquina de base de datos es una combinación de hardware, software, y firmware, que efectúa algunas o todas las funciones de manejo de la base de datos para la computadora central.

1.7 EL CICLO DE VIDA DE UN SISTEMA

El objetivo de la ingeniería del software es definir los métodos ingenieriles para construir sistemas complejos por grupos de personas. A diferencia de la programación que es una actividad personal, la ingeniería de software es una actividad de grupo. La ingeniería de software se propone definir el proceso de producción de software, es decir contestar a las siguientes preguntas:

¿ Qué es lo que sigue ? .- Definir el orden de actividades y pasos en cada actividad.

¿ Hasta cuando lo vamos a hacer ? .- Establecer criterios para progresar de un paso a otro.

Los modelos más destacados del proceso de producción del software (ciclos de vida) son los siguientes:

1. Codifica y fija (cde-and-fix)

- Proceso:

- codificar un poco.
- probar y eliminar los errores.
- repetir el proceso.

- Ventaja.- efectivo para productos pequeños.

- Desventaja.- diseño de "espaguetti", no manejable para sistemas grandes.

- Característica.- basado en inspiración personal.

2.- Cascada.- (Waterfall)

- Proceso:

- **estudio de factibilidad.**
- **análisis de requerimientos.**
- **diseño.**
- **codificar y probar módulos.**
- **integrar y probar el sistema.**
- **entregar.**
- **dar mantenimiento.**

- Ventaja.- requiere de disciplina, planeación y administración, la implementacion debe ser propuesta hasta que se entienda bien el objetivo.

- Desventaja.- es rígido, lineal, monolítico, no realista, es difícil introducir modificaciones sobre todo en la parte del mantenimiento, la cual en 20% es correctiva, otro 20% adaptativa pero en más del 50% dirigida al perfeccionamiento del sistema, no enfoca apropiadamente el diseño de familias de programas, asume una progresión relativamente uniforme de pasos de elaboración, no se ajusta al

desarrollo evolucionario y no enfoca los posibles módulos futuros de desarrollo de software.

- Característica.- enfocado a la producción de documentación.

3.- Evolutivo.- (evolutionary)

- Proceso: es un proceso incremental y repetitivo.
- haz un prototipo .
- mide qué tan lejos es de lo esperado.
- toma en cuenta las observaciones para generar el siguiente prototipo.

- Ventaja.- ideal para sistemas que no tienen bien definidos sus requerimientos.
- Desventaja.- es difícil de distinguirlo del proceso "codifica-y-fija".
- Característica.- enfocado a la producción de prototipos.

4.- Transformativo.- (transformational)

- Proceso:
- haz la especificación formal.
- conviértela automáticamente en código.

- Ventaja.- las modificaciones se hacen a nivel de especificación y no a nivel de código, ahorra tiempo de diseño, codificación y pruebas.
- Desventaja.- bueno para productos relativamente pequeños de áreas limitadas (hojas de cálculo 4GL's).
- Característica.- trabajo a nivel de especificación.

5.- Espiral.- (spiral)

- Objetivo.- minimizar los riesgos en todas las etapas del desarrollo de un sistema. Es más bien un metamodelo a todos los métodos anteriores.
- Ventaja.- toma en cuenta el riesgo y trata de minimizarlo en las siguientes etapas del proceso de desarrollo de un sistema. Permite aplicar lo mejor de los demás modelos.

- Desventaja.- no es maduro, depende de la habilidad humana para evaluar adecuadamente los riesgos y tomar las decisiones correctas, faltan técnicas y métodos para aplicar sus pasos.

1.8 NUEVAS FRONTERAS PARA APLICACIONES DE B.D.

Los sistemas manejadores de bases de datos (DBMS) son utilizados frecuentemente en aplicaciones de tipo administrativas, contables, etc. siendo raro encontrarlas en áreas como ingeniería o en ciencias exactas, no obstante ésto, no quiere decir que no sean importantes.

Todas las actividades de un proceso de ingeniería son típicamente dirigidas en una diversa gama de lugares que en la mayoría de los casos se encuentran en lugares distantes; la información está sujeta y es generada por herramientas especializadas en diversas arquitecturas de hardware. Los datos son usados en medios de almacenamiento incompatibles, en múltiples estructuras y formatos que son adecuados más por las características de la herramienta que por el uso que se le dará posteriormente a la información.

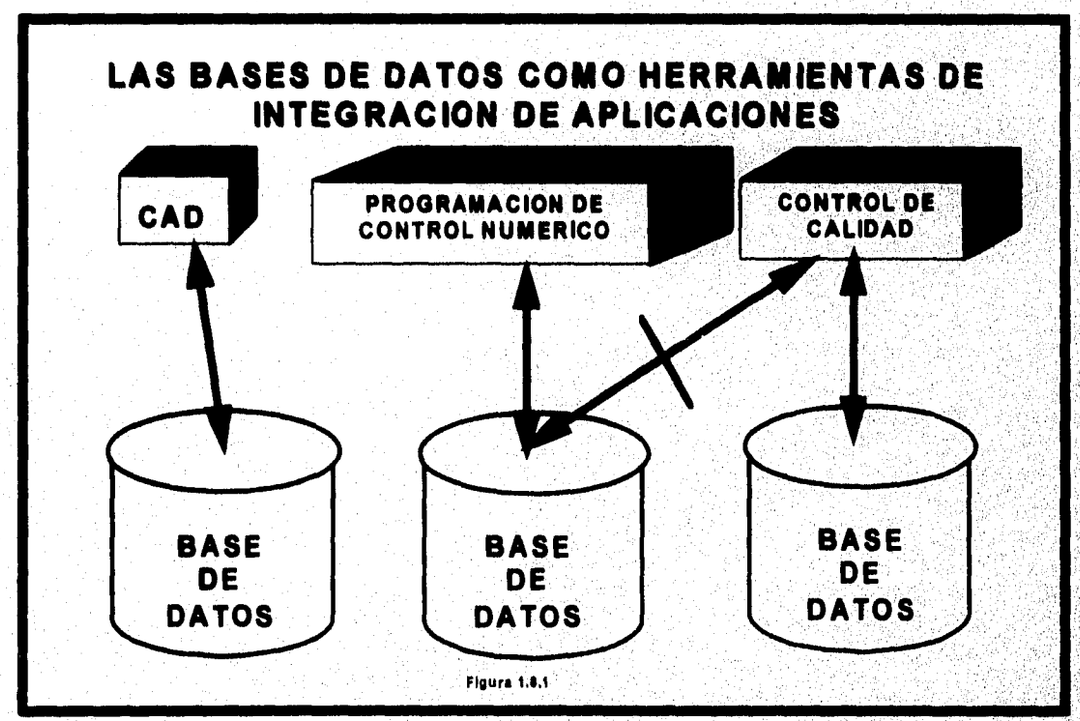


Figura 1.8.1

Las bases de datos organizan la información y cada DBMS formatea la información de acuerdo a sus procesos, **aquí el problema es claro, hay que estandarizar la información**. Uno de los principales cambios en el manejo de la información en las aplicaciones en ingeniería y en general, es la integración de diversos módulos de propósito especial en diferentes plataformas, estructuras, etc. El usuario debe ver la información de manera transparente, el problema entonces requiere sistemas abiertos, no cerrados, es importante modelar la información tal como la percibimos en el mundo real y mapearla exactamente a un modelo computacional, hacer esto es fácil con las técnicas de orientación a objetos. Cada uno de los pasos debe ser provisto de una visión de los datos en forma homogénea e integrada, en un nivel lógico el cual se pueda asociar directamente con el mundo real de la ingeniería, considerando la importancia al mismo tiempo del desempeño y economía.

1.9 ARQUITECTURAS DE LAS BASES DE DATOS (COMO HERRAMIENTAS DE INTEGRACION DE APLICACIONES)

Se analizan a continuación las diferentes arquitecturas que han evolucionado para soportar bases de datos, las cuales son resultado de las necesidades cambiantes de los sistemas de bases de datos.

Nivel 0 (AISLAMIENTO)

En el nivel 0 de integración, cada una de las aplicaciones maneja su propia información en una base de datos o sistema de archivos altamente acondicionada de acuerdo a las necesidades de la aplicación y totalmente incompatible con las demás bases de datos, de tal forma que las estructuras de información no puedan ser accedidas por cualquiera otra aplicación. Para poder integrar la información, lo único que podemos hacer es transformar la estructura de cada una de las aplicaciones en forma manual o semiautomática.

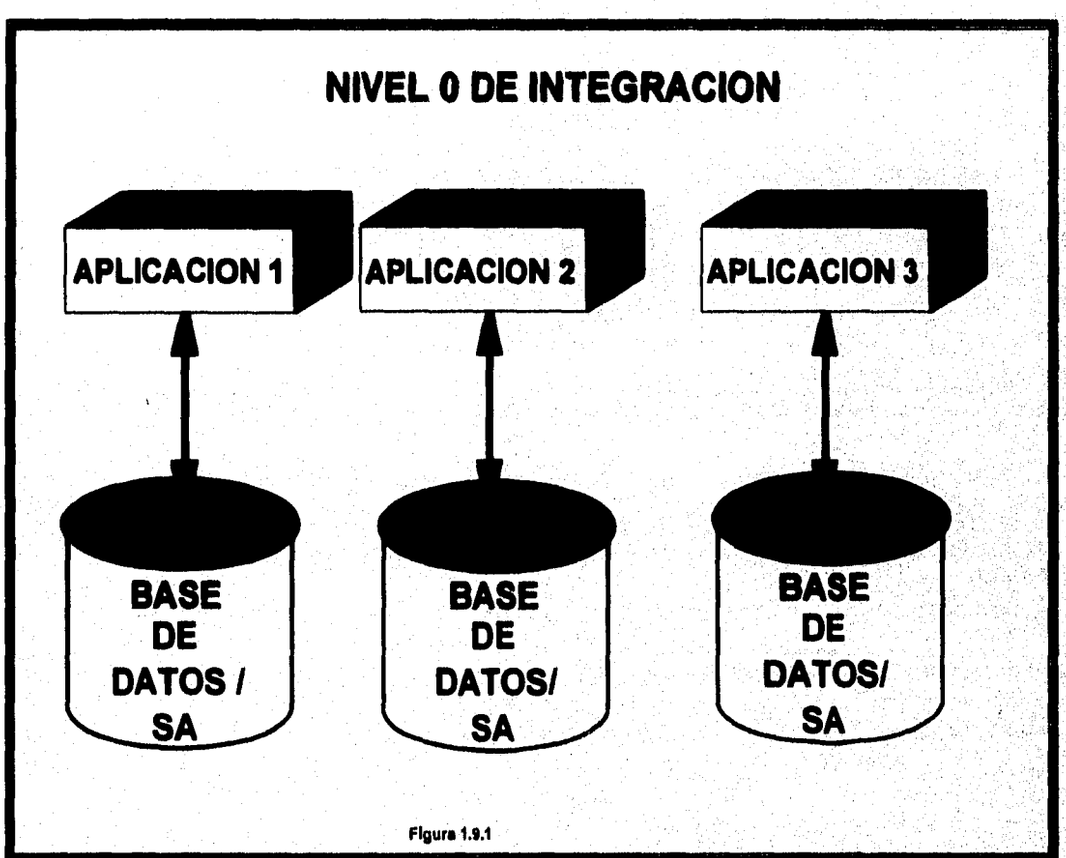


Figura 1.9.1

Nivel 1 (CONVERTIDORES)

En esta arquitectura la interfaz entre diversos módulos (sistemas de bases de datos) se realiza a través de convertidores que transforman la información y sus estructuras en el tipo de formato y estructura que requiere el módulo de otra aplicación. Desafortunadamente el número de convertidores que se necesitan para tener una interfaz entre N módulos es N*(N-1)

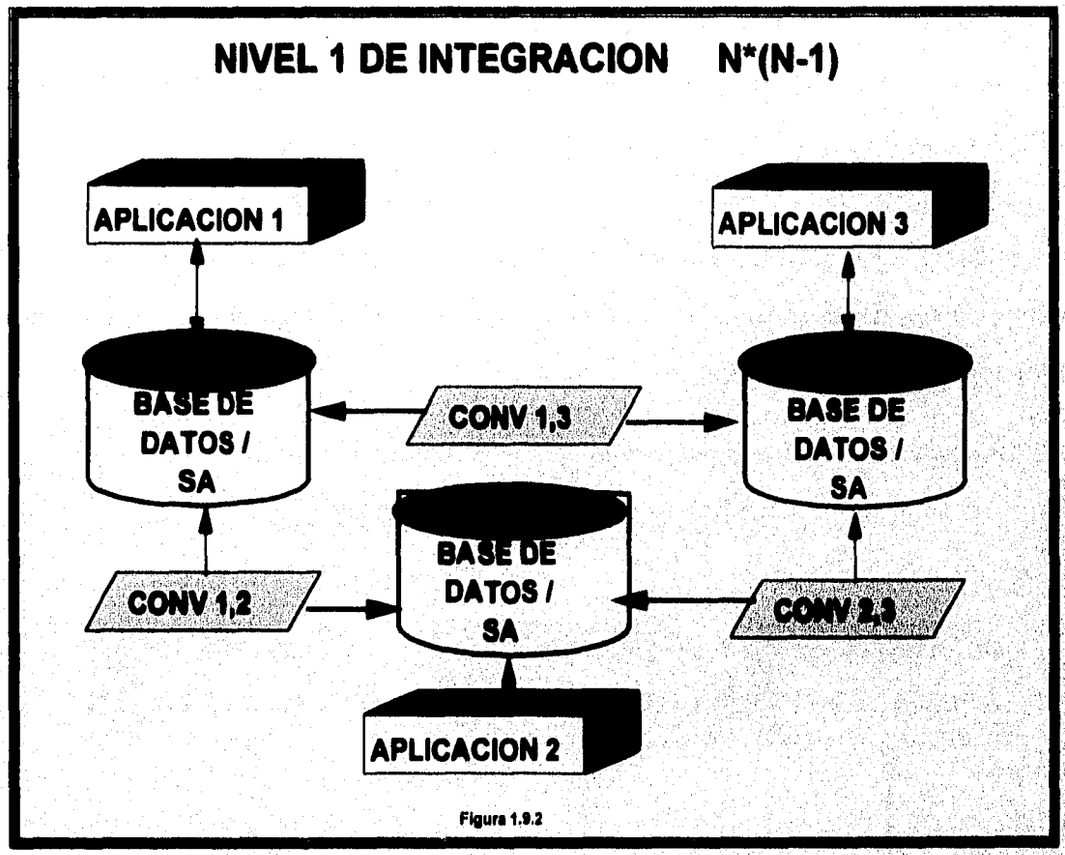


Figura 1.9.2

En la figura, cada convertidor Ci, es en realidad dos convertidores, uno en cada sentido.

Nivel 2 (FORMATO DE ARCHIVO NEUTRAL)

Esta arquitectura se conoce como FORMATO DE ARCHIVO NEUTRAL y se caracteriza porque reduce el número de convertidores requeridos, asumiendo una base de datos o sistema de archivos neutral.

Esta estructura neutral puede ser vista como un intercambiador de datos estándar. El número de convertidores requeridos en este caso es de 2*N

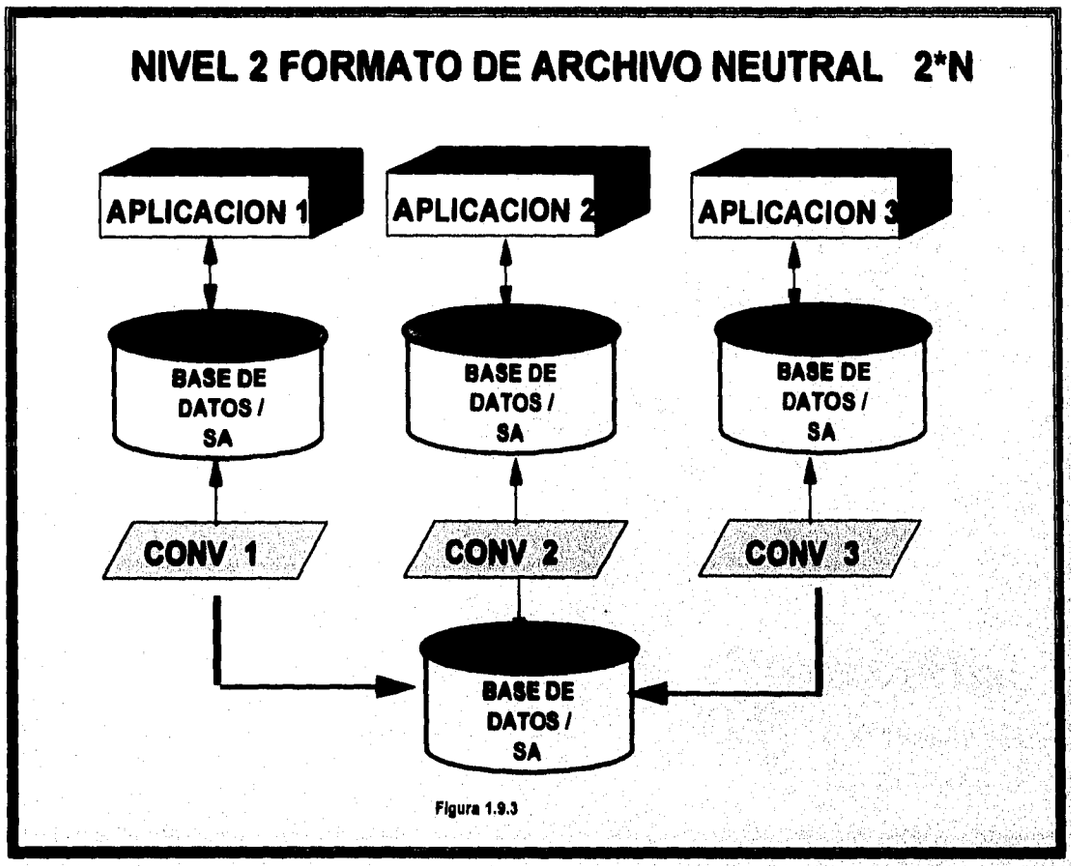


Figura 1.9.3

Nivel 3 (BASES DE DATOS CENTRALIZADAS)

Los DBMS tradicionalmente han sido considerados como uno de los medios más importantes para la integración de información, controlando su generación, uso e intercambio de información en ambientes complejos. La centralización puede ser a un nivel lógico; sin embargo, físicamente ésta puede estar distribuida en varias

computadoras, siendo la distribución transparente para los usuarios. La centralización de los datos trae consigo las siguientes ventajas:

- No son necesarias las conversiones.
- Toda la información es accesible por los usuarios autorizados.
- Es más fácil obtener la información y puede ser reutilizada.
- Se tiene una mayor consistencia de la información, con escasa redundancia evitándose las estructuras incompatibles.

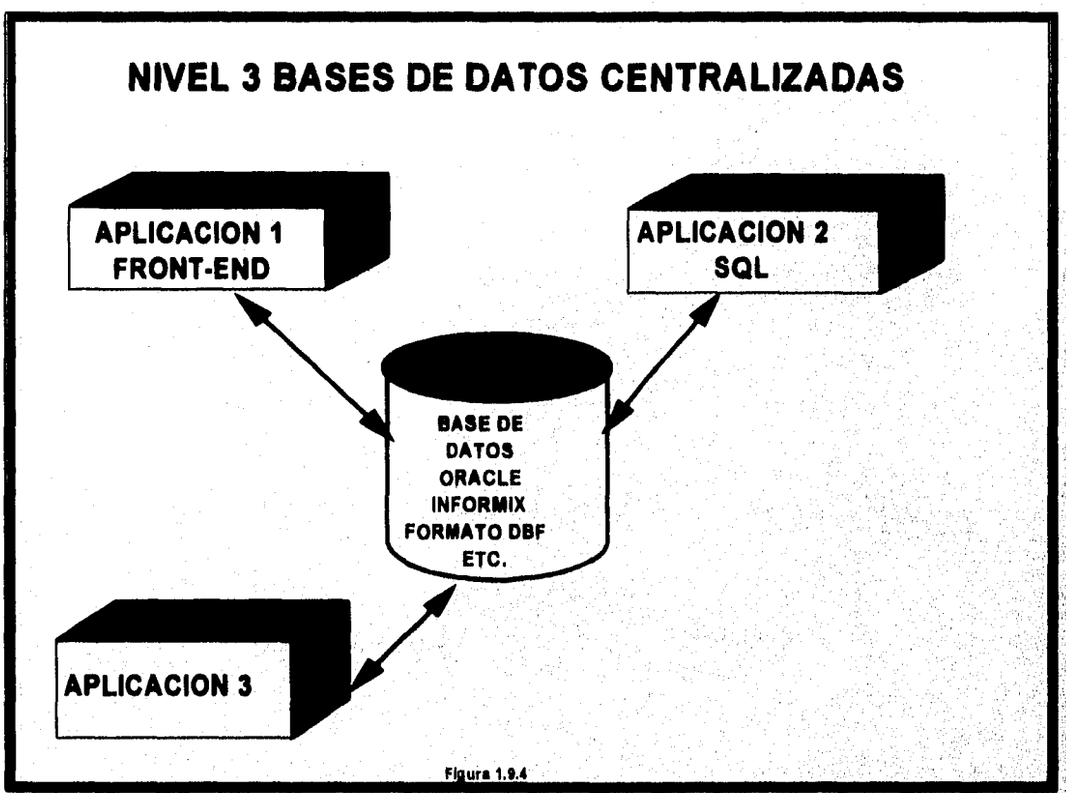


Figura 1.9.4

Ese nivel es uno de los mejores ya que cada una de las aplicaciones entiende el formato y no son necesarios convertidores, además de que este nivel se apoya en las bases de datos distribuidas.

Todos los esquemas de bases de datos reales, es decir actuales, se centran en este esquema donde la aplicación puede ser un front-end como visual basic, object vision, etc. y la base de datos ORACLE, INFORMIX, formato DBF, etc.

Nivel 4 INTEGRACION DE COMPONENTES INDEPENDIENTES.

Combinación de los niveles 2 y 3. Como se mencionó anteriormente el nivel 3 se conoce como el mejor de todos, sin embargo muchos módulos de ingeniería como CAD, CAM, CAQ, etcétera, tienen formatos altamente adecuados y difícilmente pueden ser reescritos para tener una interfaz con un DBMS. Por lo tanto, se puede combinar una solución entre los niveles 2 y 3 utilizando convertidores especializados, conocidos como APLs.

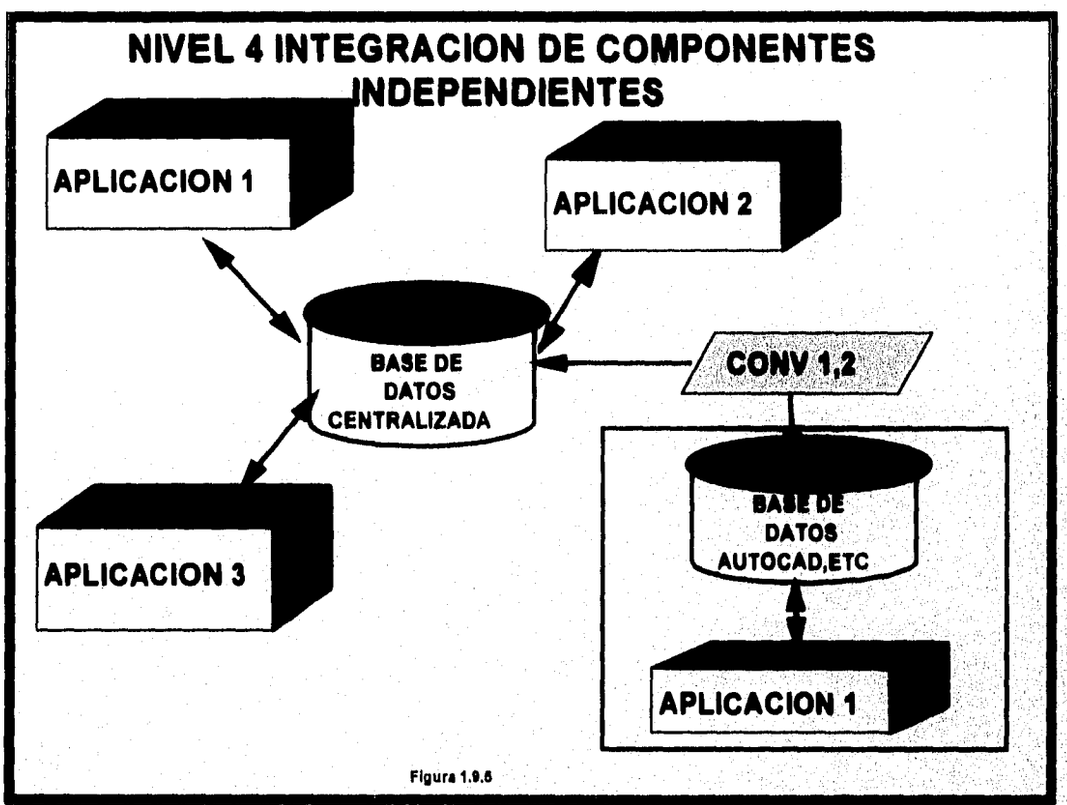


Figura 1.9.6

CAPITULO II

INTRODUCCION A LA FILOSOFIA Y LA TERMINOLOGIA DE LA ORIENTACION A OBJETOS

> **La "orientación a objetos" será la más importante de las tecnologías que surjan en los años noventa.**
>
> **Bill Gates, Presidente de Microsoft Corp.**

> **La Programación OO no es una panacea. No es algo en que todos los casos pueda servir como la cera para el suelo y el adorno para los postres. Es sencillamente mejor que las alternativas actuales.**
>
> **Deutsch 1989.**

CAPITULO II

INTRODUCCION A LA FILOSOFIA Y LA TERMINOLOGIA DE LA ORIENTACION A OBJETOS

2.1 LA FILOSOFIA DE LA ORIENTACION A OBJETOS

Después de años de obscuridad relativa, la orientación a objetos "OO" ha descendido del Olimpo académico y parece estar introduciéndose en la corriente principal de la computación comercial, su impacto se muestra en todos los ámbitos del que hacer informático; un giro hacia la orientación de objetos esta sucediendo simultáneamente a lo largo de la tecnología informática de punta, incluyendo lenguajes de programación, interfaces de usuario, bases de datos y sistemas operativos. Prueba de ello es la práctica que se percibe en las grandes casas del desarrollo de software, las cuales no sólo están inundando el mercado de productos OO, sino lo que es más significativo, están reestructurando sus productos ya existentes utilizando tecnología OO, buscando el beneficio que se deriva en términos **de flexibilidad, modularidad, reusabilidad,** en un intento por prepararse para enfrentar la dinámica del cambio y competencia comercial en estos tiempos de globalización de mercados. Poco a poco este concepto ha invadido el mercado. Los productos para desarrollo de aplicaciones más populares y más antiguos ahora están enfocados a los objetos. La OO es una de las nuevas y más poderosas tecnologías en computación. Ello no es simplemente la adición de otras características a los lenguajes de programación, sino una moderna manera de pensar acerca de los problemas, descomponiéndolos en procesos y colocando en ellos agentes autónomos (objetos), cada uno de los cuales es responsable de tareas específicas. Está tecnología ha llegado a ser una escuela importante de diseño de software que es independiente de cualquier lenguaje en particular. **La metodología ha sido desarrollada en los Estados Unidos de América y Europa (Noruega, Suecia, Alemania, etc.). Para facilitar el diseño y construcción de sistemas más robustos, amigables, confiables y adaptables al cambio,** adicionalmente estas tecnologías han

mostrado la posibilidad de reducir tiempos y costos de desarrollo y mantenimiento; **es decir, la tecnología OO se ha vuelto sinónimo de calidad y robustez.**

El término revolución industrial en el software se ha utilizado para describir el paso hacia una era en la que el software será compilado a partir de componentes de objetos reutilizables, con lo que se crearía una enorme biblioteca de componentes. Debemos pasar de una era de sistemas de software monolíticos, donde un vendedor "construye" todo un complejo producto de software, hasta una era en donde el software será ensamblado a partir de componentes y programas de software de muchos proveedores, **debemos construir el software de igual forma que el hardware es construido.**

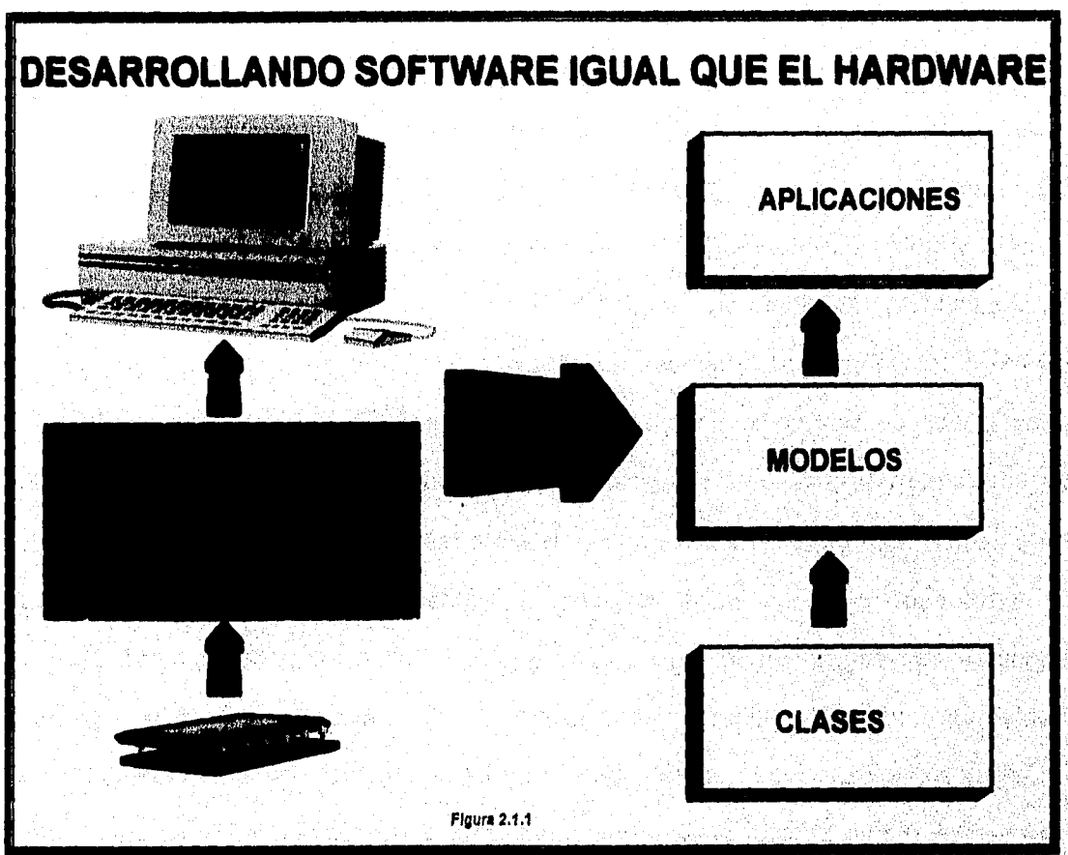


Figura 2.1.1

La programación orientada a objetos permite niveles más altos de abstracción desde el objeto a la clase hasta la biblioteca de clases y finalmente los complejos marcos estructurales de aplicación (modelos).

Los sistemas orientados a objetos han sido empleados en la comunidad investigadora por más de 20 años, estas compañías han refinado la técnica y por fin la tecnología de objetos tiene un camino fuera de los laboratorios. Compañías como IBM, ED, TEXACO, Texas Instruments, etc. han adoptado la tecnología de orientación a objetos como parte de su desarrollo profesional. Miles de programadores en la principales compañías de software comercial, ya están utilizando las herramientas de orientación a objetos.

La orientación a objetos al estar integrada en los componentes fundamentales del software, significa para la década de los años 90's, lo que la programación estructurada significó en los 70's, por lo cual **se encamina con paso firme al uso general.** Esta tecnología es una herramienta poderosa para la construcción de software que trata de resolver los clásicos problemas que existen en el desarrollo del software, el sólo concepto de tecnología de objetos nos dice la posibilidad de la reusabilidad del software, confiabilidad, rapidez, calidad, integridad, desarrollo y mantenimiento más sencillos, así como mayor nivel de automatización de bases de datos.

En general, los métodos de diseño de sistemas de bases de datos, orientados a funciones, fueron establecidos para producir diseños en lenguajes procedimentales tradicionales de alto nivel, tales como COBOL FORTRAN o Pascal. En ellos los procesos están rigurosamente separados de los datos. Los procesos en una base de datos, en una jerarquía de subprogramas, o los datos en un sistema de archivos. La siguiente figura nos muestra cómo existe la separación de los datos y las funciones (métodos) o subprogramas. **El principal paradigma de la programación orientada a funciones es el de la descomposición modular en las funciones del sistema.** como se muestra en la siguiente figura:

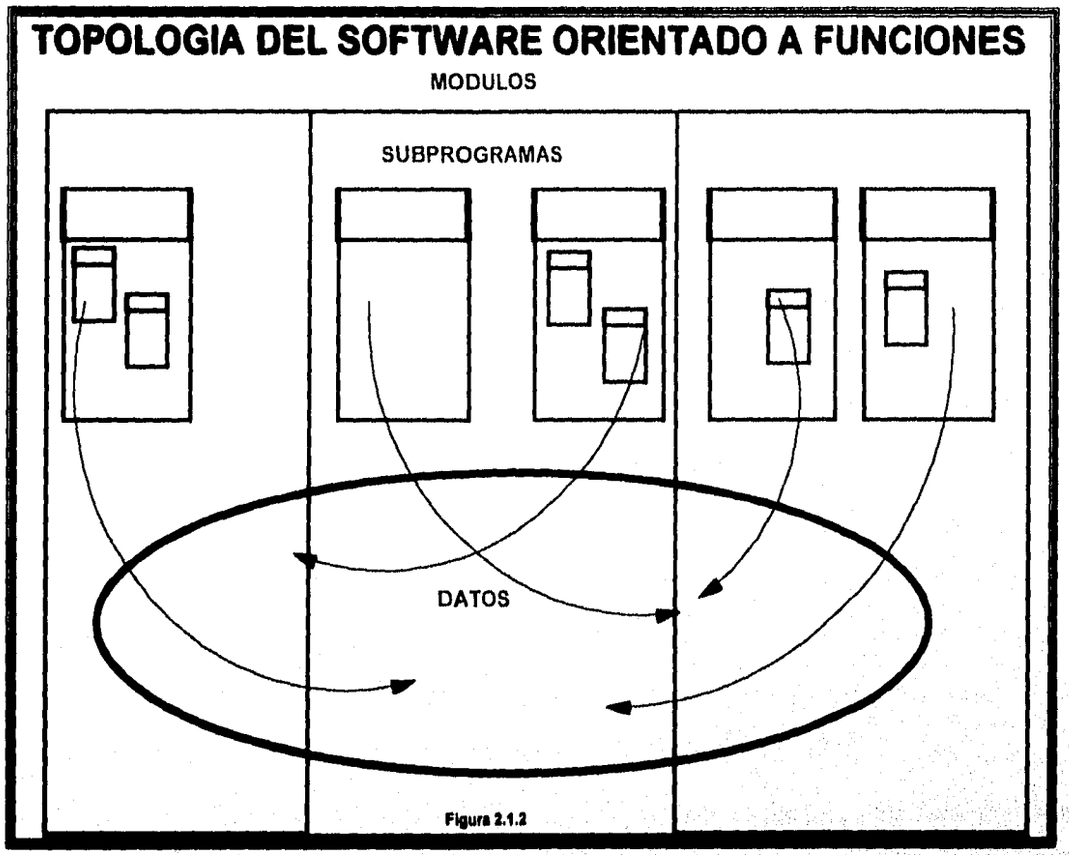


Figura 2.1.2

Los sistemas están formados por módulos y los módulos a su vez están formados por programas; y estos por subprogramas, los subprogramas están construidos por funciones.

Entre los métodos orientados a funciones, el más influyente y quizá el más empleado de todos los métodos estructurados es el asociado a los nombres de Larry Constantine, Tom de Marco y Edward Yourdon.

Los métodos de diseño orientados a objetos han sido establecidos para producir diseños para la construcción de software en los lenguajes de alto nivel más recientes, basados en objetos y orientados a objetos* tal como Ada, C++, Common Lisp Object System (CLOS), Oberon, Object Pascal, Smalltalk y Visual Basic.

* Se dice que un lenguaje es basado en objetos si soporta objetos como una de sus características; se dice que es orientado a objetos si, adicionalmente requiere que tales objetos pertenezcan a clases que puedan ser modificadas incrementalmente a través de la herencia (Wegner).

Las Técnicas orientadas a objetos permiten que el software se construya a partir de objetos de comportamiento específico. Los propios objetos se pueden construir a partir de otros, que a su vez pueden estar formados de otros objetos.

El diseño de software orientado a objetos pretende ser una técnica de construcción de aplicaciones que funcione de forma acorde a como lo hace la mente humana, en su ubicación en la realidad. El hombre conoce mediante un proceso de abstracción en el que de lo real vamos tomando sus aspectos más comunes para construir formas inteligibles por nuestro pensamiento, estas formas son los objetos. Los objetos deben contener un conjunto de propiedades que los distingan y definan, algo que les de la individualidad necesaria como para que el conocimiento del hombre pueda tratar con ellos. Sólo conocemos objetos. **"Los objetos son la formalización de lo real que hace la mente del hombre"**. Por lo tanto si la noción de objeto posee tanta importancia a la hora de explicar los mecanismos cognoscitivos del ser humano, parece lógico que en una de las actividades intelectuales del hombre, como es el diseño de software, se trate de aprovechar todo lo que esta noción puede dar de sí.

La orientación a objetos anima al desarrollador de sistemas a concentrarse en los temas importantes e ignorar el resto. La OO es el fundamento de las tareas administrativas distribuidas a través de la empresa y la funcionalidad a escala del grupo de trabajo a nivel empresarial. La tecnología habilitará la interacción de aplicaciones por medio del uso de un intermediario de solicitudes de objetos, que envía los mensajes entre las aplicaciones para recuperar datos administrativos. La posibilidad de representar a los dispositivos como objetos, nos permite que se agrupen en clases con características administrativas comunes. Concuerda con la forma en que en una empresa se realiza el trabajo .

La tecnología orientada a objetos permite por primera vez vincular múltiples eventos, procesos y respuestas de manera que manejemos la resolución del problema mucho más rápido que sin la tecnología, con una mínima intervención del usuario.

En los métodos orientados a objetos, el concepto subyacente es que los sistemas de software deberían modelarse como colecciones de objetos cooperantes que procesan a los objetos individuales, como instancias de una clase. Cada objeto mantiene en una cápsula sus procesos y sus datos.

Coad y Jourdon definen a la orientación a objetos como:

Orientación a objetos = objetos + clasificación + herencia + comunicación

Grady Booch resume la diferencia fundamental entre programación procedimental (o basada en procedimientos) y programación orientada a objetos, de la siguiente forma:

"Lea las especificaciones del software que desea construir, subraye los verbos si persigue un código procedimental, o los nombres si su objetivo es un programa orientado a objetos. (BOOCH 1989)". La descripción de Booch implica que el paradigma orientado a objetos, con su énfasis puesto en los objetos en vez de los procesos, es tan diferente de la orientación procedimental como los nombres lo son respecto a los verbos.

La idea fundamental de la orientación a objetos es que el usuario no debería tener que batallar con construcciones orientadas al computador tales como registros y campos, sino más bien debería poder manejar objetos (y operaciones) que se asemejen más a sus equivalentes del mundo real, es decir **la idea principal es elevar el nivel de abstracción para construir sistemas artificiales a partir de los sistemas naturales.**

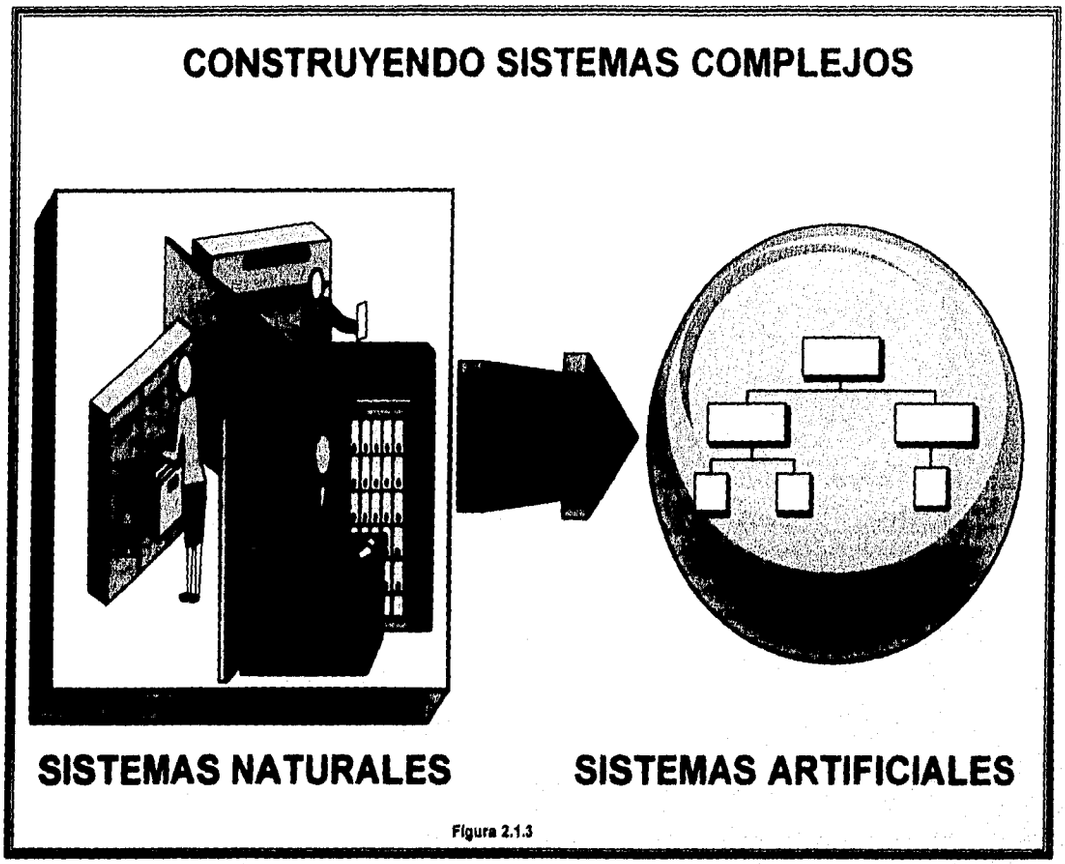


Figura 2.1.3

Mecanismos básicos

La orientación a objetos proporciona un nuevo paradigma para la construcción de software. En este nuevo paradigma los objetos, las clases y las variables de instancia (o modelo) son los pilares, mientras que los métodos, los mensajes y la herencia producen los mecanismos primarios. Todos los sistemas que merecen la descripción de orientación a objetos contienen estos mecanismos esenciales, aunque los mecanismos pueden no estar realizados (o denominados) exactamente de la misma forma.

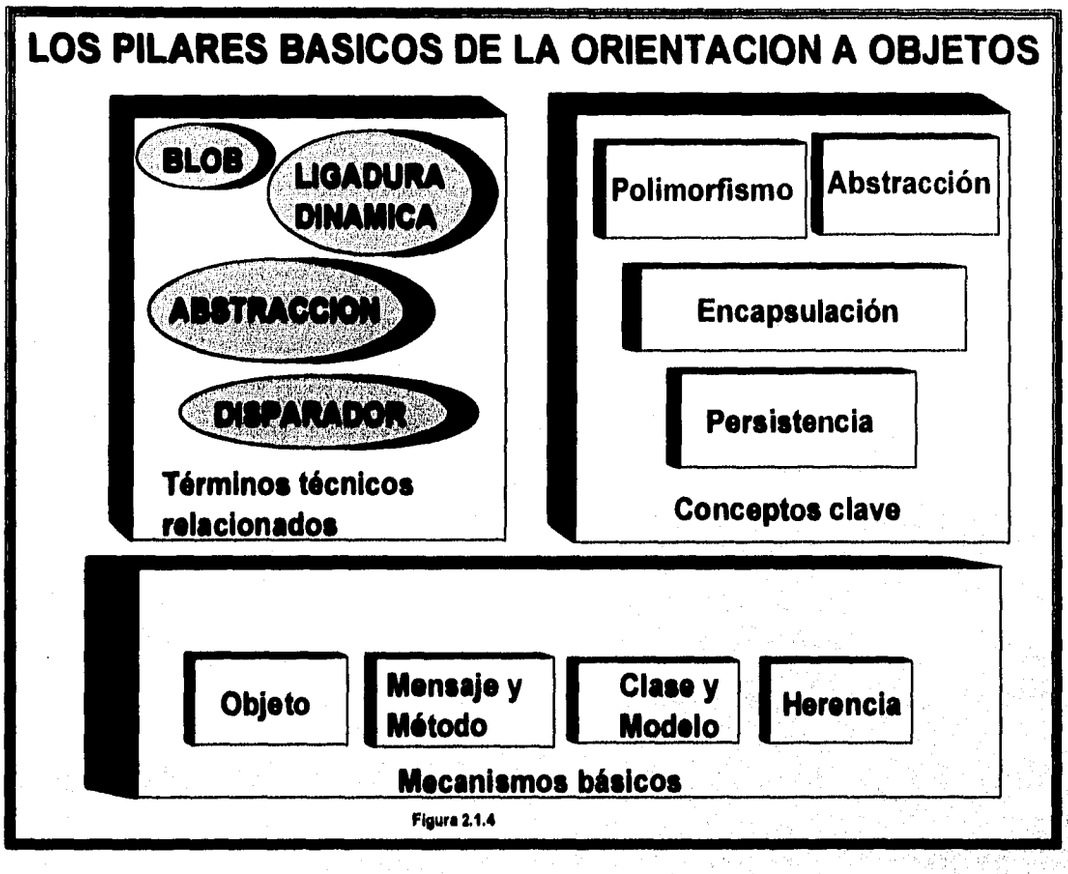


Figura 2.1.4

2.2 OBJETOS

En el mundo físico en el que nos movemos, todo puede ser considerado como objeto: un automóvil, una persona o un circuito integrado. Los objetos tienen características como su color o tamaño, exhiben, asimismo, un cierto comportamiento. Por ejemplo, corren o cambian de estado en respuesta a una serie de estímulos. Estos objetos del mundo real se pueden utilizar una y otra vez sin que haya necesidad de volverlos a diseñar. Muchos circuitos integrados son artículos estándar baratos y pueden adquirirse en una tienda. El hecho de que estas piezas se puedan adquirir, permite que los diseñadores se puedan concentrar en resolver los problemas que se les presentan sin tener que volver a diseñar sus herramientas más elementales como microprocesadores, contadores, multiplexores, ampificadores operacionales, etc.

La definición más común de un objeto es:

Objeto = Estado + Métodos

Donde:

- Estado.- Consiste en un conjunto de atributos, que describen la condición actual del objeto, los cuales varían dinámicamente, y no son directamente accesibles y/o modificables desde afuera.
- Métodos.- Son como "puertos" de interacción con el objeto, que responden a mensajes, es la única forma de interactuar con el objeto.

El concepto de un objeto es simple y poderoso, los objetos forman módulos de software ideales, debido a que pueden definirse y mantenerse independientemente, formando cada uno, un universo autocontenido. Todo lo que un objeto conoce está expresado en sus variables y todo lo que puede hacer está expresado en sus métodos.

Un objeto es una entidad computacional activa, consistente en una colección de campos de información (su estructura interna) y un conjunto de aplicaciones que actúa sobre esa información (su interfaz). Un objeto es en cierta manera análogo a un módulo al que se ha agregado una jerarquía; bajo esta analogía, un módulo es un registro al que se le ha adicionado un conjunto de operaciones que actúa sobre sus campos de información .

El concepto de un objeto en un lenguaje de programación provee un mecanismo para modelar el mundo real, las entidades en el problema pueden ser representadas como objetos y las operaciones realizadas por estas entidades es factible ajustarlas como funciones de los objetos. También es posible definir a un objeto como una unidad de información contenida en sí misma donde los datos son protegidos y manipulados. En la programación orientada a objetos, **el movimiento se inicia por la transmisión de un mensaje a un objeto responsable de realizar cierta labor.** El aviso lleva codificado el requerimiento de la acción.

Un programa tradicional consta de procedimientos y datos, mientras que un programa orientado a objetos consta solamente de objetos que contienen tanto los

procedimientos como los datos. **El Software orientado a objetos tiene otra filosofía muy diferente a las tradicionales**, la figura siguiente muestra su topología.

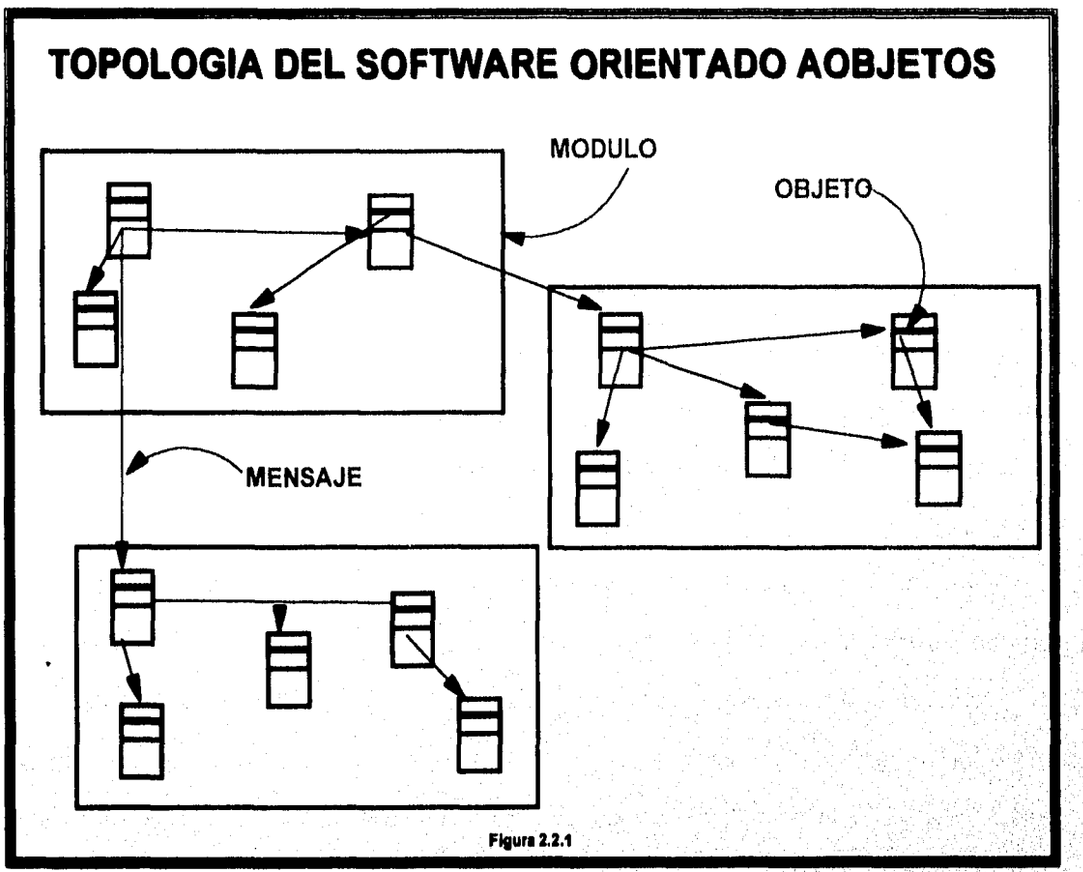


Figura 2.2.1

Un objeto es una entidad definida que posee identidad propia (para dos objetos, es posible saber si es el mismo o son distintos), valores (que pueden ser otros objetos) asociados con propiedades (también llamadas atributos o ranuras), y métodos (que a menudo necesitan argumentos) propios del objeto.

Se denomina objeto a todo lo que es capaz de admitir un predicado cualquiera, todo lo que puede ser sujeto de un juicio. Se trata de la noción más general posible, ya que no importa si lo concebido existe o no: basta con que se pueda pensar y decir algo de ello. Desde nuestra perspectiva un objeto será cualquier entidad individual discreta que pertenece al menos a una clase que esta dotada de propiedades, que es

capaz de realizar determinados procesos y cuya simple apariencia (interfaz) oculta la complejidad de su construcción.

La primera especie de Objetos que se ofrece a nuestra consideración es la de los objetos concretos. Son los que ordinariamente se nos dan en la experiencia sensible, que poseen especialidad y temporalidad y que interactúan a través de acciones también concretas. Son los Objetos del mundo real.

De los objetos concretos hay que distinguir con todo rigor los objetos conceptuales. Son objetos creados por la mente humana totalmente ajenos tanto a la espacialidad como a la temporalidad, como los objetos matemáticos (números, variables figuras) y los objetos lógicos (conceptos, proposiciones, teorías). Los objetos conceptuales no actúan de ninguna manera, ni entre sí ni con objetos concretos: las relaciones existentes entre ellos son a la vez conceptuales.

ODBMS Versant define a un objeto como la construcción de un software que puede encapsular los datos y hacer armar relaciones entre los datos y el código.

Los objetos pueden contener una clase de datos, incluyendo valores, imágenes, sonido, documentos o bien referencias de otros objetos. Los objetos pueden ser asociados con una clase de código usando las características computacionales de un lenguaje interface como C++, o Smalltalk que tienen diferentes caminos para definir y usar los objetos también tienen pequeñas diferencias en la terminología en general.

Los objetos, ya sean reales, virtuales o bien imágenes virtuales de objetos reales:

- Tienen interior y exterior.
- Están formados por objetos más simples.
- Tienen estructura interna.
- Tienen un repertorio de operaciones (conducta).
- Interaccionan entre si.

Los objetos son generalmente reconocibles unos de otros por sus formas, apariencias y conducta externa.

Su exterior está fuera de nuestro alcance, existen objetos que son iguales o parecidos, aunque a veces los podemos distinguir por su estado interno.

En la metodología de objetos, la principal preocupación es cómo representarlos en la computadora (estos son los objetos virtuales)

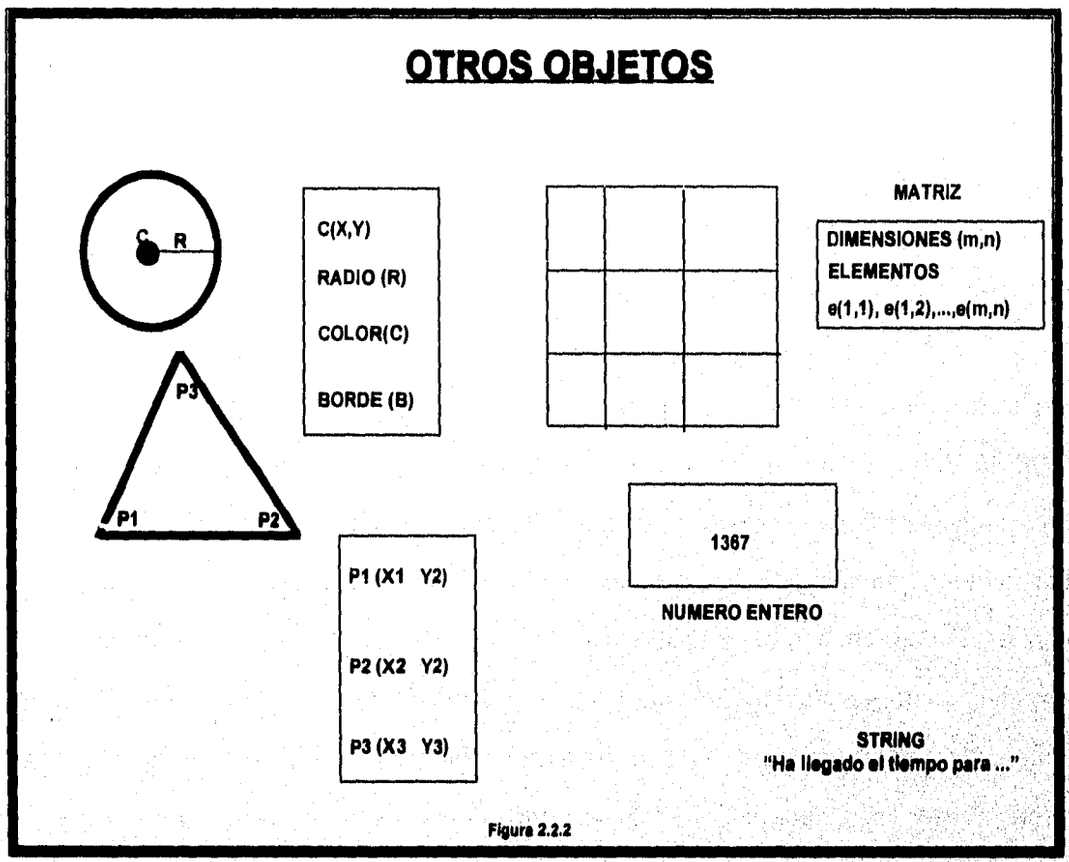


Figura 2.2.2

Podemos pensar en la representación interna de un circulo como una tabla de cinco números (X,Y,R,C,B), a partir de ellas podemos obtener la impresión de que se trata de un circulo, en virtud de las operaciones asociadas al objeto.

En especial, la operación dibujada asociada al círculo, es la que nos produce su forma; las otras permiten moverlo, cambiarle de color, hacerlo más pequeño, tal como esperaríamos hacerlo con un circulo real.

Lo mismo sucede con otros objetos tales como rectángulos, matrices, vectores, ventanas, etc.

Un objeto es parte de una aplicación totalmente terminada y transportable de forma que pueda usarse de forma similar por otras aplicaciones. La labor de ensamblar objetos es más sencilla que la de desarrollar sistemas partiendo sólo de un lenguaje de programación. Los objetos son pues, entidades dinámicas que existen como áreas de memoria durante la ejecución de un programa.

OBJETOS COMPUESTOS

Al objeto que contiene otros objetos se le conoce como objeto compuesto. Cómo se muestra en la siguiente figura, en donde podemos apreciar que un apuntador contiene la dirección del objeto que literalmente "contiene" otros objetos; por conveniencia, la referencia del apuntador se almacena en una variable llamada identificador de objetos **ID**. Esta técnica brinda dos principales ventajas:

1. El "contenido" del objeto puede cambiar de tamaño y composición, lo cual facilita el mantenimiento en sistemas complejos.
2. El "contenido" del objeto es libre y puede participar o ser parte de numerosos objetos. Esto es vital para capturar información del mundo real.

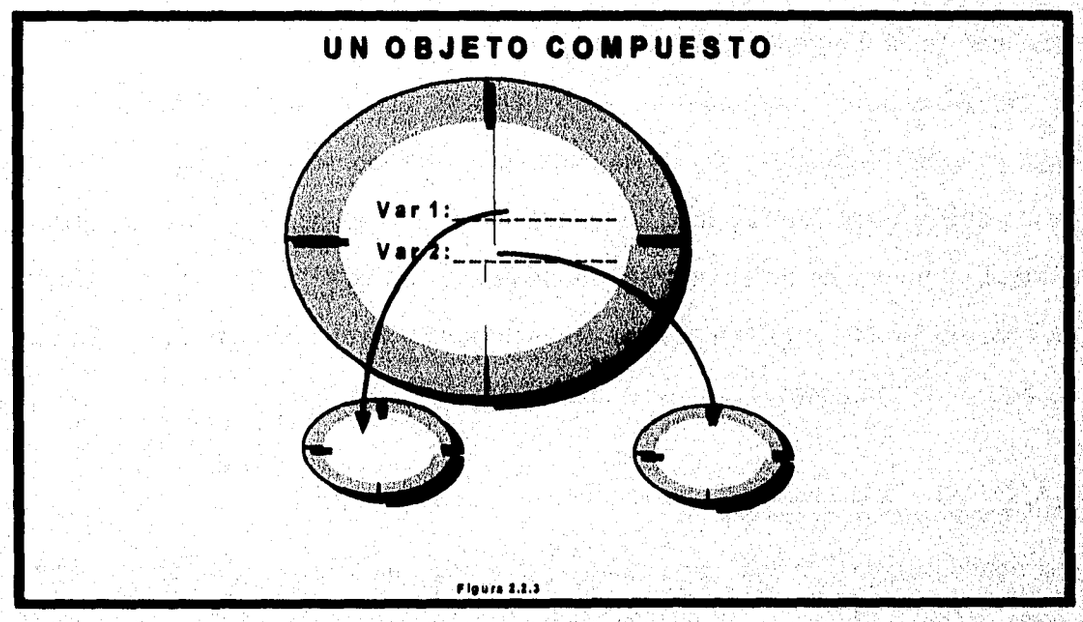


Figura 2.2.3

Un objeto Complejo, es un objeto construido a partir de:

- objetos atómicos (integer,reals,string, boolean,...).
- usando constructores: (Tuple x o [...], set * o { ...}, list ...).
- Relaciones (set of tuples).
- Registros jerárquicos (tuples of tuples).
- Relaciones Non First Normal Form NF2,etc.
- Nuevas estructuras por aplicaciones ortogonales de los constructores (por ejemplo: Sets of sets).

EL CICLO DE VIDA DE UN OBJETO

Los objetos antes de nacer están quiesentes y su ciclo de vida es el siguiente:

Xo (estado inicial)

x1(estado final)

xf (estado extinto).

Las condiciones de frontera de un objeto van del estado quiesente al inicio y del final al extinto.

Ejecución de un programa orientado a objetos

Cuando se ejecuta un programa orientado a objetos, ocurren tres sucesos: en primer lugar se crean los objetos cuando se necesitan; segundo, los mensajes se mueven de un objeto a otro (o desde el usuario a un objeto) a medida que el programa procesa internamente información o responde a la entrada del usuario, los objetos reciben, interpretan y responden a dichos mensajes; y tercero, se borran los objetos cuando ya no son necesarios y se recupera memoria.

MENSAJES Y PROTOCOLOS

Los objetos del mundo real se afectan de diversas formas entre sí: crear, agregar, mover, enviar, doblar, etc. esta tremenda variedad genera un problema interesante, como es posible representar todas estas interacciones en el software. Una solución

elegante a este problema lo es el mensaje, es decir, **a diferencia de los elementos de datos pasivos en los sistemas tradicionales, los objetos tienen la posibilidad de actuar.** La acción sucede cuando un objeto recibe un mensaje, que es una simple solicitud que hace un, objeto a otro objeto, solicitándole al segundo objeto que se comporte de alguna forma o ejecute un(os) método(s).

Por convención el objeto que transporta el mensaje es llamado "sender" transmisor y el objeto que recibe el mensaje se conoce como "receiver" receptor.

a = sender y b = receiver.

- a "conoce" b.
- le manda un mensaje:
- le avisa de algo.
- le pide que haga algo.
- le pide información sobre su estado.
- El mensaje tiene argumentos u otros objetos.
- puede recibir un valor como respuesta.

El protocolo de objeto es el conjunto de mensajes que el objeto puede recibir, interpretar y responder. La respuesta de un objeto a un mensaje es la realización de una operación (un método). Normalmente las operaciones que realizan los objetos incluyen una o varias de las siguientes acciones:

Enviar mensajes a otros objetos, cambiar su propio estado interno, **disparar** la creación de otros objetos, o la destrucción propia de los otros.

Por ejemplo, el protocolo para un icono puede constar de mensajes invocados por la pulsación del botón cuando el usuario localiza un puntero sobre un icono.

En la siguiente figura vemos cómo los objetos interactúan a través de mensajes.

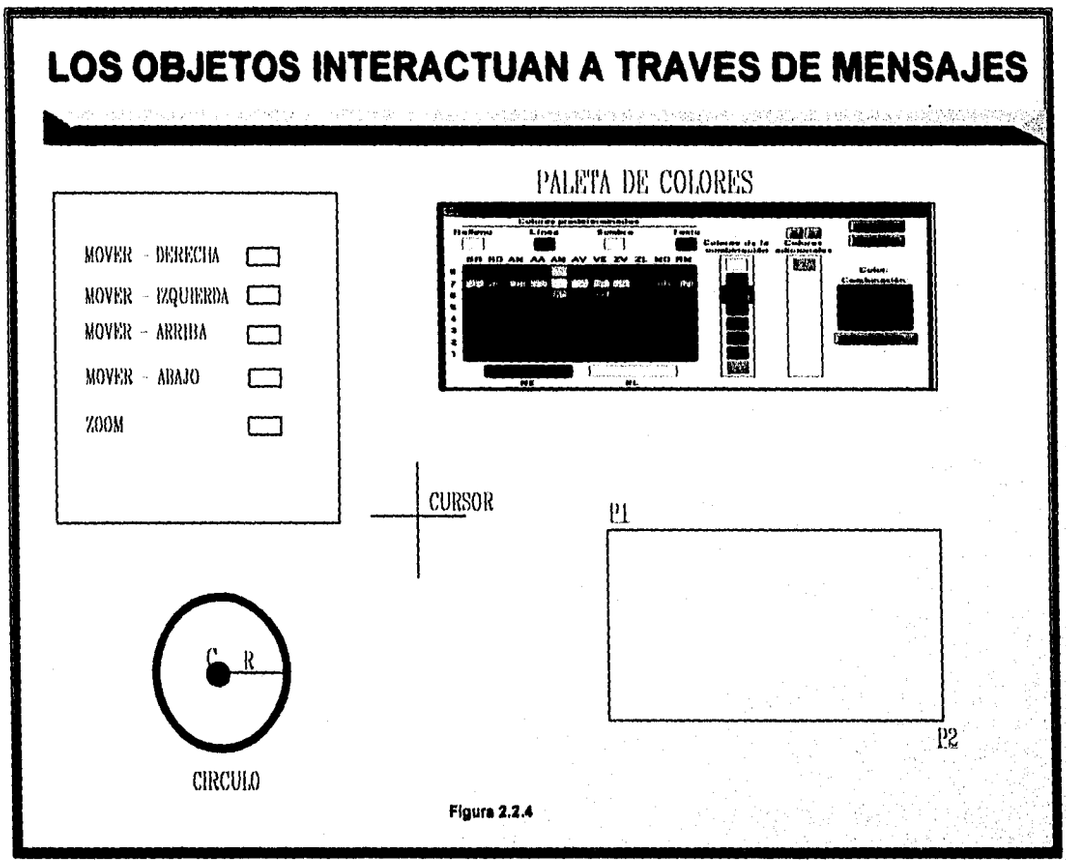


Figura 2.2.4

A través del objeto menú podemos enviar un mensaje al círculo para que se mueva o cambie de tamaño.

Podemos también usar la paleta para seleccionar un color lo cual produce un mensaje al menú que éste envía al rectángulo y éste cambia de color.

Podemos ir más lejos y pensar en un objeto mouse que envía mensajes para controlar el movimiento del objeto cursor, que a su vez envía mensajes a los objetos menú, paleta, círculo y rectángulo. El objeto menú está formado por una especie de rectángulo que contiene otros objetos (ternas formadas por leyendas, botones e instrucciones); Las instrucciones están ligadas a las leyendas y producen mensajes específicos cuando el botón respectivo es activado mediante un mensaje que recibe del objeto cursor, cuando éste se encuentra dentro del botón y recibe la señal enter del teclado o del mouse.

Estructuralmente un mensaje consiste de tres partes: la identificación del receptor, el método que ejecutará el receptor, y la información especial (los paramentros) que

requiere el método para ejecutarse, esta ultima información por lo general viene estructurada en forma de parámetros, aunque bien puede ser el nombre de un valor de dato o identificación de un objeto, los parámetros son opcionales y típicamente están compuestos de cero a cinco. La siguiente figura muestra el ejemplo de un mensaje:

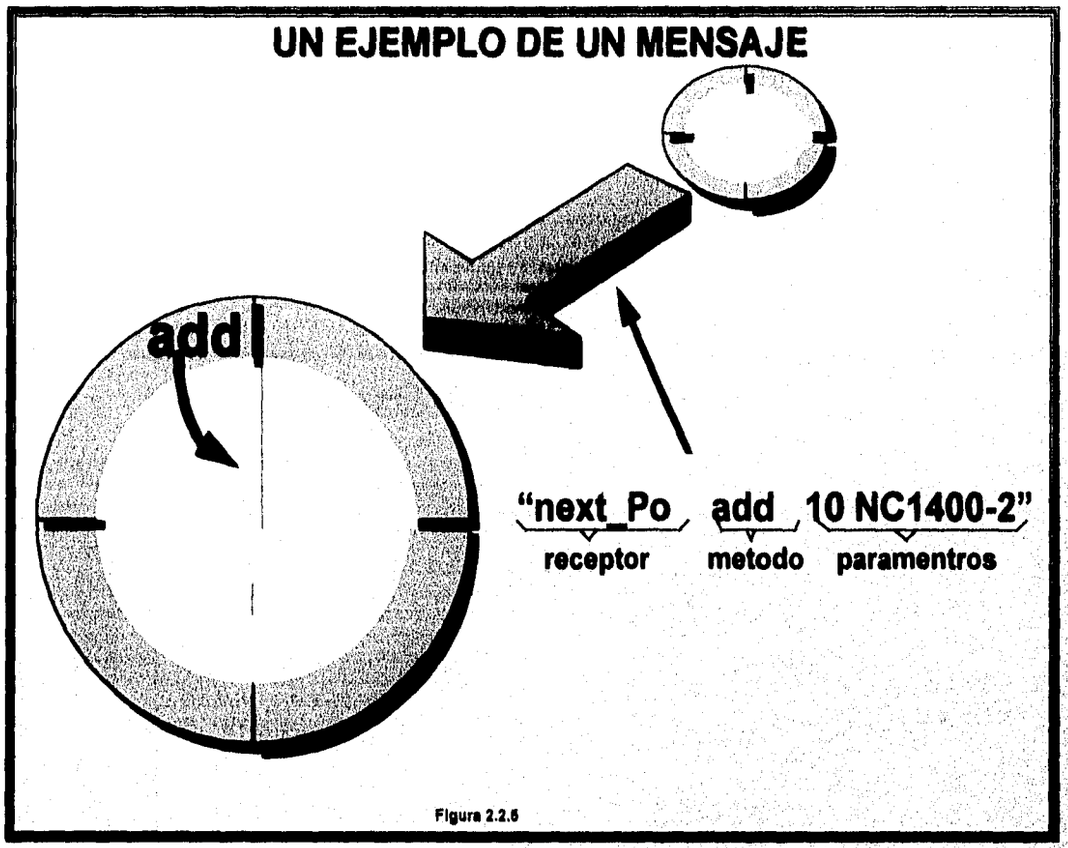


Figura 2.2.6

El "receiver" receptor es next_PO.

El método es add, con los requerimientos de ordenes de compra y agregar otra línea al ítem itself.

Los parámetros son 10 y NC1400-2, con la llamada suma 10 copias al inventario e ítem conoce que es NC1400-2.

METODOS

Un método es un algoritmo que contiene los pasos necesarios que han de ejecutarse como respuesta a un mensaje, es decir son procedimientos (código) que residen en el objeto y determinan cómo actúa el objeto cuando recibe un mensaje. Además, las variables modelo o de instancia almacenan información o datos locales en el objeto (por lo que **a un objeto se le conoce también como instancia**). Los métodos se ejecutan en respuesta a mensajes y manipulan los valores de las variables modelo. De hecho, **los métodos proporcionan el único mecanismo para cambiar los valores de las variables modelo.** Los métodos pueden enviar también mensajes a otros objetos solicitando acción o información, la siguiente figura muestra la anatomía de un objeto.

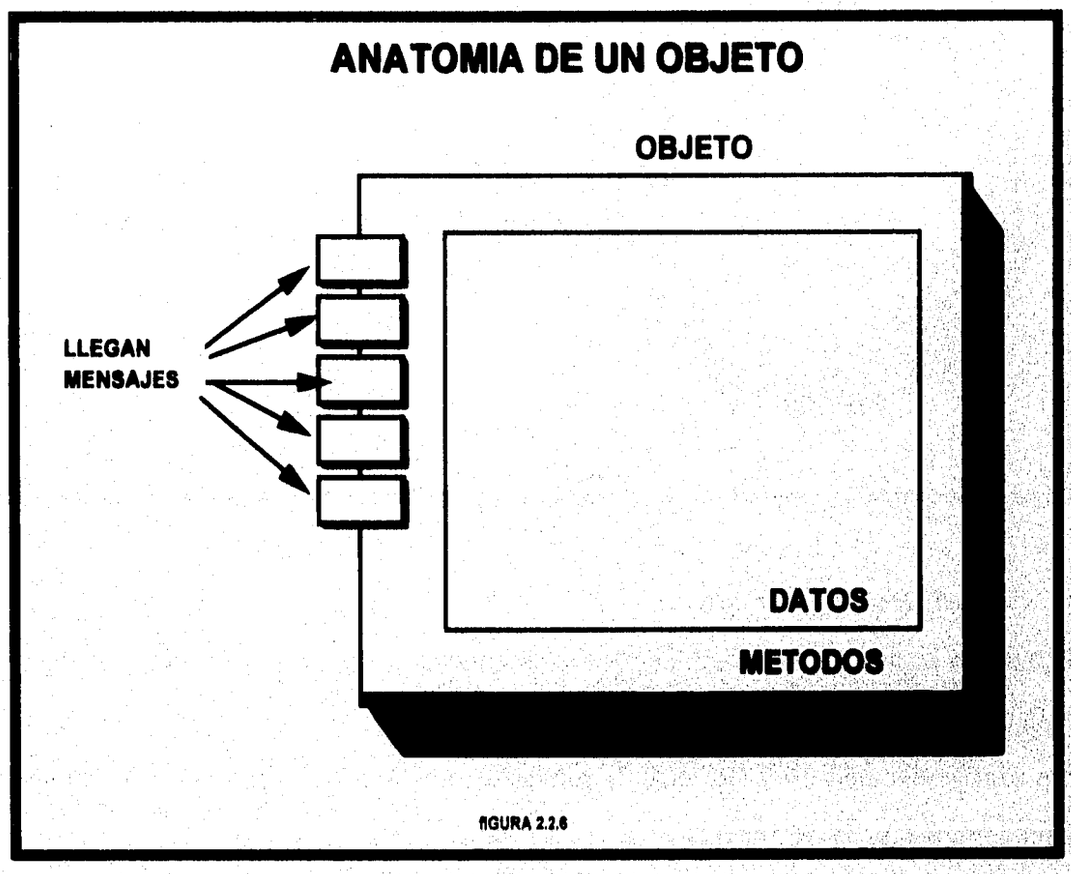


FIGURA 2.2.6

METODOS

Un método es un algoritmo que contiene los pasos necesarios que han de ejecutarse como respuesta a un mensaje, es decir son procedimientos (código) que residen en el objeto y determinan cómo actúa el objeto cuando recibe un mensaje. Además, las variables modelo o de instancia almacenan información o datos locales en el objeto (por lo que **a un objeto se le conoce también como instancia**). Los métodos se ejecutan en respuesta a mensajes y manipulan los valores de las variables modelo. De hecho, **los métodos proporcionan el único mecanismo para cambiar los valores de las variables modelo.** Los métodos pueden enviar también mensajes a otros objetos solicitando acción o información, la siguiente figura muestra la anatomía de un objeto.

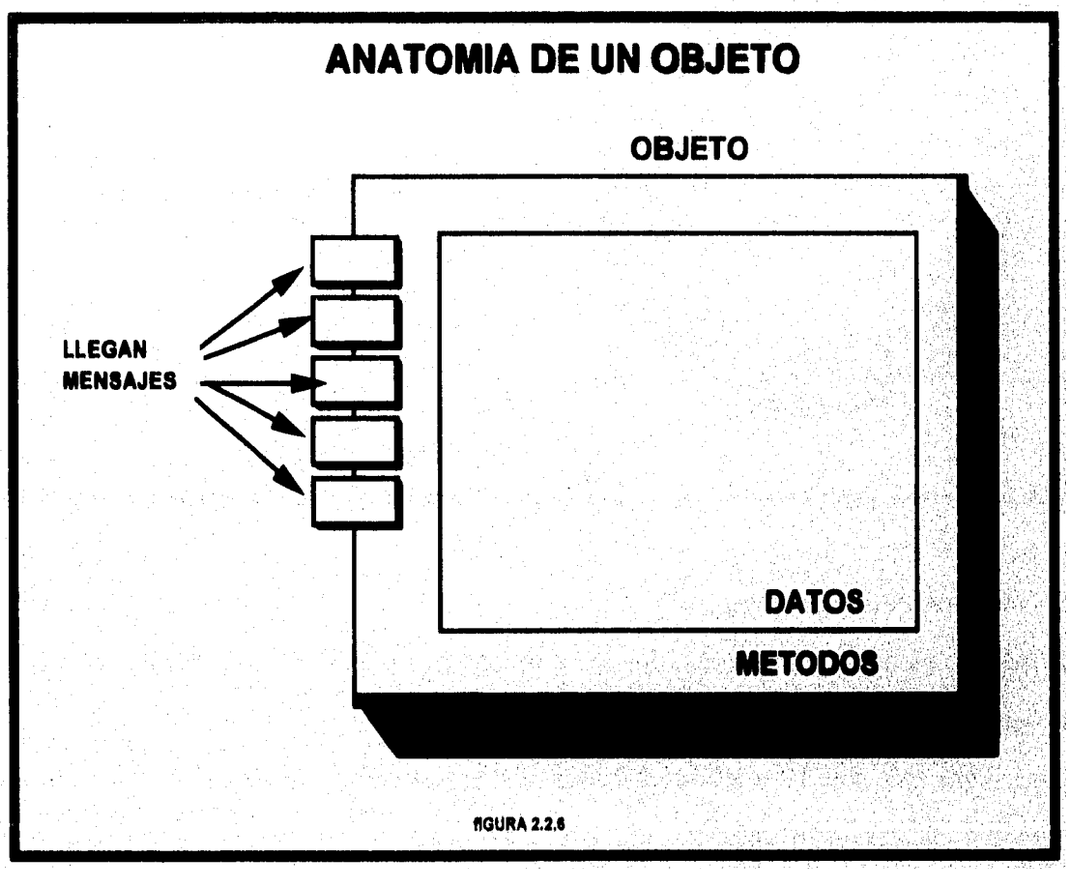


FIGURA 2.2.6

Los mensajes que recibe el objeto son los únicos conductos que conectan al objeto con el mundo exterior.

2.3 CLASES E INSTANCIAS

En raras ocasiones un sistema involucra sólo un objeto de cada tipo. Es más común requerir más de uno. Una clase es un prototipo que define los métodos y variables que serán incluidas en un tipo de objeto en particular. Las descripciones de los métodos y variables que los soportan se describen sólo una vez, en la definición de la clase. **Una clase es pues un conjunto o colección de objetos que tiene ciertas ranuras (atributos o variables) y métodos (procedimientos) en común.** De hecho, las propiedades se asocian (cuando se definen) a una clase, pero "las posee" cada objeto (individuo*) en particular.

Muchos objetos diferentes pueden actuar de formas muy similares. Una clase es una descripción de un conjunto de objetos casi idénticos. Una clase consta de métodos y datos que resumen las características comunes de un conjunto de objetos.

Una clase es un objeto que tiene como propiedad el poder crear otros objetos, dotándoles de una peculiar herencia. Todos los objetos de una clase reciben de ésta sus rasgos comunes.

Por ejemplo, Ciudad Capital describe todas las ciudades del mundo que son capital de un país. Se dice que los objetos descritos por la clase son instancias de ésta, es decir, una instancia o ejemplar de un objeto es el resultado de la creación de descriptores (o conjunto de datos) que representan dicho objeto.

Por ejemplo Madrid es una instancia de la clase ciudad Capital

* Un individuo es un objeto que no es una clase. Es decir no es una colección. Ejemplo : La Paz es un individuo miembro de la clase CIUDAD, pero CIUDAD no es un individuo.

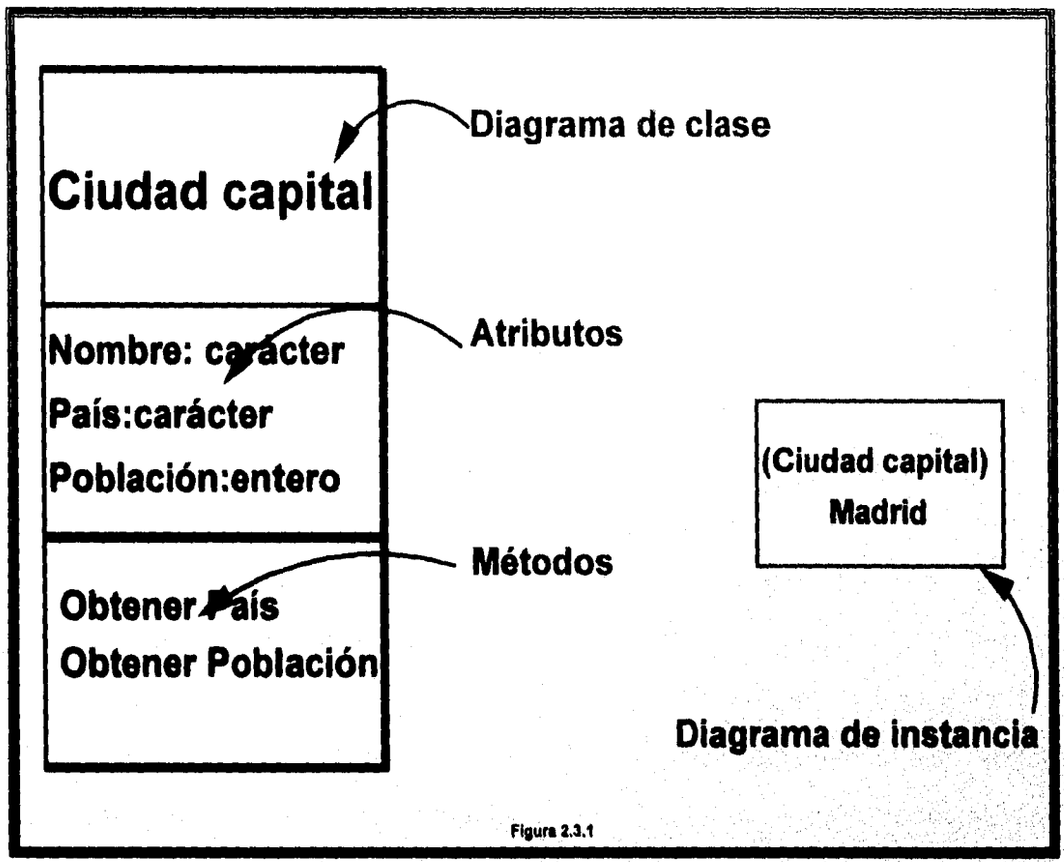


Figura 2.3.1

Las instancias de un objeto (registros) pueden ser objetos matemáticos, texto, dispositivos de la máquina, descripciones de la máquina, descripciones de cualquier objeto real, código (en lenguaje de máquina o pseudocódigo), una instancia por lo tanto se puede: crear, borrar o destruir.

Una clase es entonces una descripción de un conjunto de objetos casi idénticos, una clase consta de métodos y datos que resumen las características comunes de un conjunto de objetos. La posibilidad de abstraer métodos y descripciones de datos comunes de un conjunto de objetos y almacenarlos en una clase es esencial para la potencia de la orientación a objetos.

Definir una clase es pues situar código reutilizable en un depósito común, evitando así la redundancia del código al evitar escribir una y otra vez el mismo código. En otras palabras, las clases contienen los anteproyectos para crear objetos.

Finalmente, la definición de una clase ayuda a clarificar la definición de un objeto. Por ejemplo un sistema de inventarios podría tener un sin número de almacenes, sería extremadamente ineficiente el definir los mismos métodos para cada objeto que represente un almacén, por lo que los autores de Simula ofrecieron la opción elegante de la Clase.

Un objeto es una instancia de una clase

Los objetos se crean cuando se recibe un mensaje solicitando creación por la clase padre. El nuevo objeto toma sus métodos y datos de su clase padre, creándose así una instancia de la clase padre. **Las variables de clase tienen valores almacenados en una clase; las variables de instancia tienen valores asociados únicamente con cada instancia u objeto creado a partir de una clase**, por ejemplo consideré como podría implementarse una aplicación de procesamiento de textos en forma orientada a objetos.

Los objetos que pertenecen a una clase se denominan instancias de la clase, y contienen tan sólo los valores particulares para las variables, compartiendo el código de los métodos.

Los métodos de la clase están ligados a la clase y los métodos de la instancia están ligados a un objeto especifico, la metaclase es la clase de los objetos clase

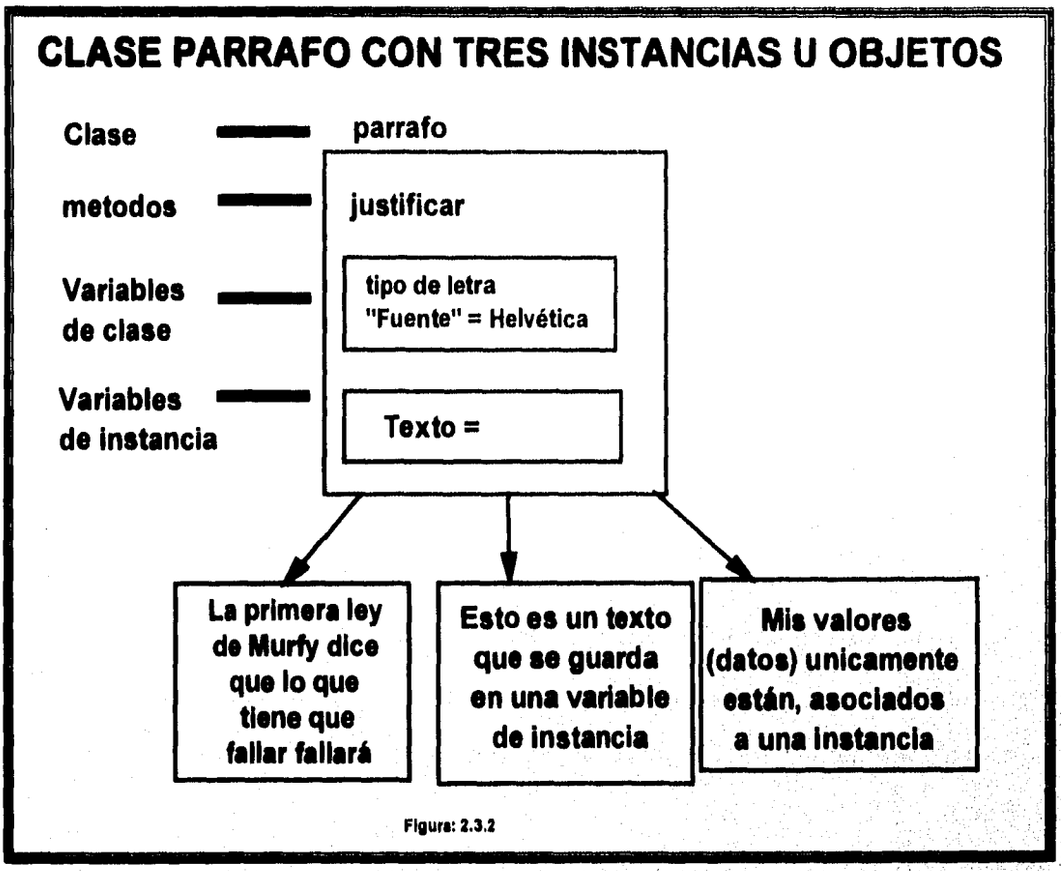


Figura: 2.3.2

ODBMS Versant define a una clase como la definición de un objeto con atributos y métodos.

Desde el punto de vista de los lenguajes de programación, las clases son plantillas estáticas que existen solamente en el cuerpo del código fuente de un programa,

Las clases no existen en forma natural, nosotros las definimos, las operaciones son servicios que las clases dan a otras clases, Las instancias de una clase existan o no deben tener un identificador que los haga únicos.

Una Clase es pues la abstracción de un conjunto de objetos con características en común donde se especifican estas características:

- **atributos**
- **métodos**

Una clase es como una fábrica de objetos, ya que la definición de los atributos se dá en la clase, pero los atributos sólo existen en las instancias de la clase. De igual

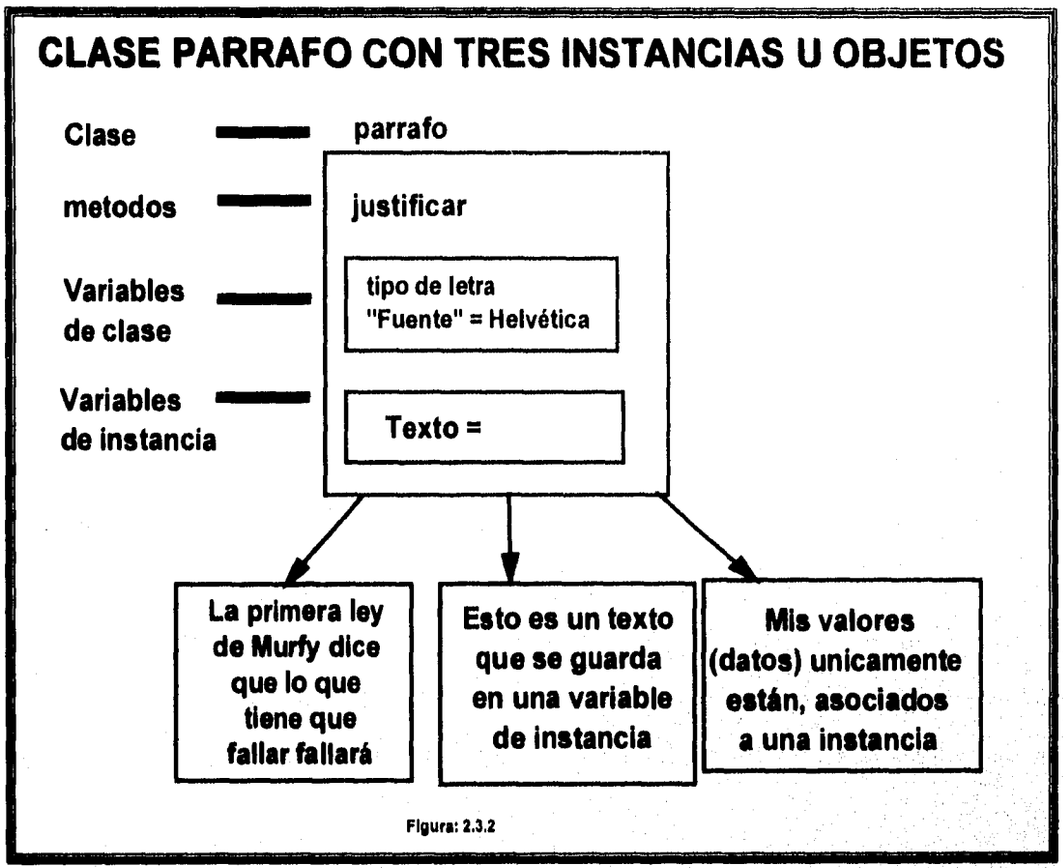


Figura: 2.3.2

ODBMS Versant define a una clase como la definición de un objeto con atributos y métodos.

Desde el punto de vista de los lenguajes de programación, las clases son plantillas estáticas que existen solamente en el cuerpo del código fuente de un programa,

Las clases no existen en forma natural, nosotros las definimos, las operaciones son servicios que las clases dan a otras clases, Las instancias de una clase existan o no deben tener un identificador que los haga únicos.

Una Clase es pues la abstracción de un conjunto de objetos con características en común donde se especifican estas características:

- **atributos**
- **métodos**

Una clase es como una fábrica de objetos, ya que la definición de los atributos se dá en la clase, pero los atributos sólo existen en las instancias de la clase. De igual

forma los métodos sólo existen en la clase, aunque los métodos son válidos para todas las instancias de la clase.

Visto de otra forma, cada objeto de la clase tiene entonces sus propios atributos (variables de instancia) y métodos para el acceso (conceptualmente).

ATRIBUTO

Una dupla >nombre de variable, tipo de datos>.

Donde: nombre de variable designa una propiedad y tipo de datos es el conjunto de los valores que la variable puede asumir.

Los atributos representan en el modelo las propiedades de los objetos del mundo real y se implementan como campos.

OPERACION

Una tripla <nombre de función, argumentos, resultados> donde nombre de función designa un proceso, argumentos es una tupla de valores (posiblemente vacía) requeridos por la función y resultados otra tupla que contiene los valores resultantes. Las operaciones representan en el modelo los procesos de los objetos del mundo real y se implementan como métodos. Cuando en una clase se especifican diferentes métodos para la implementación de una misma operación, se dice que ésta es polimórfica.

IDENTIDAD

Lo que hace la identidad de un objeto es un conjunto de atributos que lo hace único. La identidad representa en el modelo la individualidad de los objetos del mundo real y se implementa como llave.

RELACIONES ENTRE INSTANCIAS

Existen dos formas para poder relacionar las instancias de las clases:

-Asociación (débil).

-Agregación (Asociación fuerte).

Una asociación es una relación entre las instancias de dos o más clases que describe un conjunto de ligas con una estructura y una semántica común. Es decir, una asociación es una colección de ligas que son acoplamientos entre instancias que se están acoplando, ejemplos:

Liga: Ser esposa de.

Una agregación: En un proyector hay una serie de partes que lo componen, el foco es un foco de la clase de focos de proyectores y el brazo es una instancia de los brazos. Cada instancia de un proyector tiene una instancia de la clase de proyectores.

La asociación fuerte o débil es una forma de relacionar las instancias de las clases.

Una agregación es una forma especial de asociación, entre un todo y sus partes, en la cual el todo está compuesto de partes.

RELACIONES ENTRE CLASES

El objeto pertenece por lo menos a una clase. Las partes pueden relacionarse solamente por generalización, que es la relación entre una clase ancestral y una o más versiones refinadas descendientes de ella.

Generalización: Nosotros como la clase de los humanos de acuerdo a Darwin descendemos de la clase de los primates, entonces las características de los primates son heredadas por nosotros, pero un gorila y un hombre es una relación entre las clases más no entre las instancias.

La generalización puede ser simple o múltiple, según la clase descendiente desciende de una o varias clases ancestrales en forma directa.

Las clases descendientes reutilizan automáticamente las características (atributos y operaciones) de toda su progenie.

La relación entre clases es la parte del diseño orientado a objetos que nos permite definir dichas relaciones basándonos en el conocimiento sobre el problema en cuestión y las relaciones se dividen básicamente en dos tipos:

1. **Relaciones de uso.** Estas se dan sí, una clase necesita de objetos de otra clase para realizar sus actividades.
2. **Relaciones de herencia.** La relación de herencia entre dos clases se da cuando una clase comparte el estado y el comportamiento con otra clase más general añadiendo o modificando algunas cosas.

2.4 HERENCIA Y GESTALT

Herencia es una palabra que conocemos, aún sin saber nada del paradigma de la orientación a objetos. Si una persona hereda el color de pelo (pelirrojo digamos) de su abuelo, entonces ese color pelirrojo es de ella, no necesita teñírselo, si Pedro hereda la altura de su abuelo, no necesita hacer nada adicional para ser alto, ¡ya es alto!. En la orientación a objetos herencia significa que podemos crear un objeto y su código asociado (encapsular) y después crear otro objeto e indicarle al sistema que este nuevo objeto heredará las características de otro, lo único que tendríamos que hacer es darle al nuevo objeto características adicionales (gestalt) que no haya heredado.

Es decir, la herencia es un mecanismo por medio del cual una clase de objetos puede definirse como un caso especial de una clase más general. Casos especiales de una clase se conocen como subclases de esa clase; la clase más general de las subclases se conoce como la superclase de las clases especializadas.

Frecuentemente las asociaciones débiles de objetos poseen atributos y aún clases que pueden ser considerados como una gestalt, es decir, como el surgimiento de nuevas cualidades. Nosotros heredamos de los primates una serie de características. En cambio, las agregaciones (asociaciones fuertes) combinan la propagación de las características de las partes en el todo (herencia) y la emergencia cualitativa (gestalt). De modo similar, en la generalización las clases heredan implícitamente las características de sus ancestros, al mismo tiempo que hacen explícita la gestalt. Herencia y Gestalt son conceptos complementarios, de modo que el conjunto de

todas las características de una clase es la unión de sus características heredadas y sus características Gestálticas.

La ganancia o cualitativamente nuevo es Gestall, y herencia lo heredado. Al asociar las clases surgen atributos que no están en ninguna de las clases que se están asociando y así surge el Gestall.

"Los lenguajes de programación deberían poder heredar una parte y no todo", ya que en el mundo real se hereda una parte, no todo.

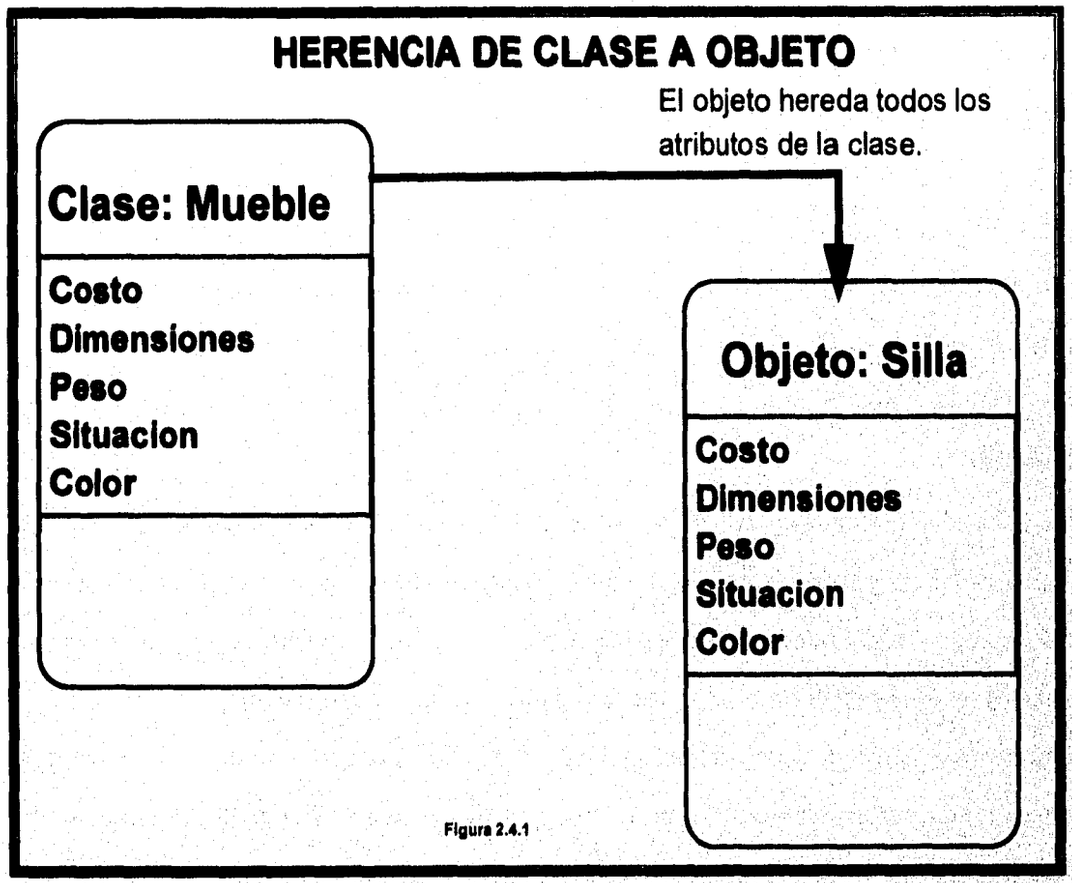


Figura 2.4.1

Las variables de instancia (campos) de una clase son análogos a los campos de registros en lenguajes como en Pascal, pueden utilizarse registros con variantes; en C estructuras con uniones. Estos mecanismos permiten que los registros tengan campos especiales, ejemplo: registros de empleados con campos especiales para los administrativos o los académicos y otros para el personal de intendencia. El

programador es responsable del mantenimiento de una etiqueta que permita distinguir entre las variantes, asimismo, el programador es responsable de cual de la variedad de empleados se tiene para que las acciones tomadas sean las adecuadas. Estos registros con variantes proporcionan una especie de subtipos. Es decir, el registro para empleados representa empleados en general, mientras que las variantes nos distinguen de un empleado-administrativo, empleado-académico, etc. Un empleado-administrativo es un subtipo de empleado en el sentido de que todo lo que se pueda hacer con empleados (métodos) es igualmente válido para los empleados-administrativos. Pero también habrá operaciones (métodos) que se aplican exclusivamente a los empleados -administrativos más no a un empleado cualquiera.

Los lenguajes de programación orientados a objetos ofrecen subtipos en una forma más elegante. El programador define una clase (empleado General) para cada subtipo y simplemente declara que estas subclases (Administrativo, Académico, etc.) heredan los métodos y variables de instancia (campos) definidos para empleados de otra Superclase, además dentro de la subclase Administrativo pueden definirse variables de instancia y métodos relevantes solamente para esta subclase de empleados-administrativos.

Las dificultades presentadas por el manejo de registros con variantes, junto con las pérdidas de productividad y la falta de confiabilidad del software, pueden ser eliminadas gracias a la herencia, el manejo de la etiqueta se convierte en responsabilidad del lenguaje; los esfuerzos requeridos para referirse a los campos de la parte variante del registro ya no son necesarios; enunciados de estilo case distribuidos a lo largo de todo un sistema para especializar la manipulación de variantes, desaparecen; el alto costo de agregar un subtipo más en el cual se incurre debido a que el conocimiento de todos los subtipos está disperso sobre todo un sistema, disminuye casi por completo y no sufre el crecimiento exponencial que se tiene al agregar subtipos de subtipos.

La herencia sencilla en C++ se efectúa solamente a través de la relación subclase-subclase-...-subclase-miembro, en donde una clase hereda a sus subclases todas sus propiedades. Finalmente, el individuo hereda de su clase inmediata (la clase más próxima a él, pero ésta ya contiene todas las propiedades necesarias) las propiedades (ranuras) que él poseerá.

Diferentes interpretaciones semánticas de la herencia:

- **Substitución:** Un empleado es una persona porque en todo momento un empleado puede substituirse por una persona.
- **Inclusión:** Un empleado es una persona porque el conjunto de los empleados es un subconjunto del conjunto de las personas.
- **Especialización:** Un empleado es una persona porque tienen todas las características de una persona, además de otras adicionales que lo hacen específico.
- **Restricción:** Un empleado es una persona porque satisface ciertos criterios.

2.5 JERARQUIAS DE CLASES

La herencia entre clases puede extenderse a cualquier grado. El resultado es una estructura arboresente conocida como jerarquía de clases. **La invención de la jerarquía de clases es el verdadero genio de la tecnología de Orientación a Objetos.** ya que el conocimiento humano se estructura de esa manera, descansando en conceptos generales y refinando en casos cada vez más especializados.

El concepto OO de jerarquía de clases es el siguiente: **si la clase B es una subclase de A, todo caso de B será automáticamente un caso de A, pero lo opuesto no se cumple**, por ejemplo podríamos tener una clase llamada EMPLEADO y otra clase llamada PROGRAMADOR, la cual es una subclase de EMPLEADO (todo programador es un empleado, pero no viceversa), PROGRAMADOR a la vez podría tener una subclase llamada PROGRAMADOR_EN_SMALLTALK (todo programador en SMALLTALK es un programador, pero no viceversa) y así sucesivamente.

La herencia sencilla en C++ se efectúa solamente a través de la relación subclase-subclase-...-subclase-miembro, en donde una clase hereda a sus subclases todas sus propiedades. Finalmente, el individuo hereda de su clase inmediata (la clase más próxima a él, pero ésta ya contiene todas las propiedades necesarias) las propiedades (ranuras) que él poseerá.

Diferentes interpretaciones semánticas de la herencia:

- **Substitución:** Un empleado es una persona porque en todo momento un empleado puede substituirse por una persona.
- **Inclusión:** Un empleado es una persona porque el conjunto de los empleados es un subconjunto del conjunto de las personas.
- **Especialización:** Un empleado es una persona porque tienen todas las características de una persona, además de otras adicionales que lo hacen específico.
- **Restricción:** Un empleado es una persona porque satisface ciertos criterios.

2.5 JERARQUIAS DE CLASES

La herencia entre clases puede extenderse a cualquier grado. El resultado es una estructura arboresente conocida como jerarquía de clases. **La invención de la jerarquía de clases es el verdadero genio de la tecnología de Orientación a Objetos.** ya que el conocimiento humano se estructura de esa manera, descansando en conceptos generales y refinando en casos cada vez más especializados.

El concepto OO de jerarquía de clases es el siguiente: **si la clase B es una subclase de A, todo caso de B será automáticamente un caso de A, pero lo opuesto no se cumple**, por ejemplo podríamos tener una clase llamada EMPLEADO y otra clase llamada PROGRAMADOR, la cual es una subclase de EMPLEADO (todo programador es un empleado, pero no viceversa), PROGRAMADOR a la vez podría tener una subclase llamada PROGRAMADOR_EN_SMALLTALK (todo programador en SMALLTALK es un programador, pero no viceversa) y así sucesivamente.

Sea B una subclase de A. En ese caso B heredará de manera automática ciertas características de A. Existen dos tipos de herencia :

Herencia Estructural. B heredará automática todas las variables de caso de A, por ejemplo, todo PROGRAMADOR tiene un SALARIO; sin embargo, B podría tener sus propias variables de caso adicionales, no compartidas por A; así los PROGRAMADORES podrían tener lenguajes (de programación), cosa que no tienen los EMPLEADOs en general.

Herencia de Comportamiento. B heredará en forma automática todos los métodos aplicables a A. Por ejemplo, los PROGRAMADORES pueden ser promovidos, porque todo EMPLEADO puede ser promovido, sin embargo, B podría tener sus propios métodos adicionales no aplicables; los PROGRAMADORES podrían tener un método AGREGAR_LENGUAJE (correspondiente a agregar un nuevo lenguaje de programación al conjunto de lenguajes que consideramos domina el programador en cuestión), en tanto los EMPLEADOS en general no posean tal método.

También podría ser posible ampliar (o aún modificar por completo la definición de un método aplicable a la clase A, de modo que realice operaciones adicionales (o totalmente distintas) cuando se aplique a un objeto de la clase B, esta posibilidad se conoce como **sobrecarga.**

Ejemplo General:

Un objeto del mundo real: La cosa sobre la que está ahora mismo sentado - una silla. **silla** es un miembro (o instancia) de una clase de objetos mucho mayor que denominamos **mueble.** Se puede asociar un conjunto de atributos genéricos a cada objeto de la clase **mueble.** Por ejemplo, todo mueble tiene: **precio, dimensiones, peso, situación y color,** entre muchos atributos posibles, que se aplican estemos hablando de una mesa o una silla, de un sofá o de una cómoda. Dado que silla es un miembro de la clase **mueble,** hereda todos los atributos definidos por la clase en la figura se muestra esquemáticamente este concepto.

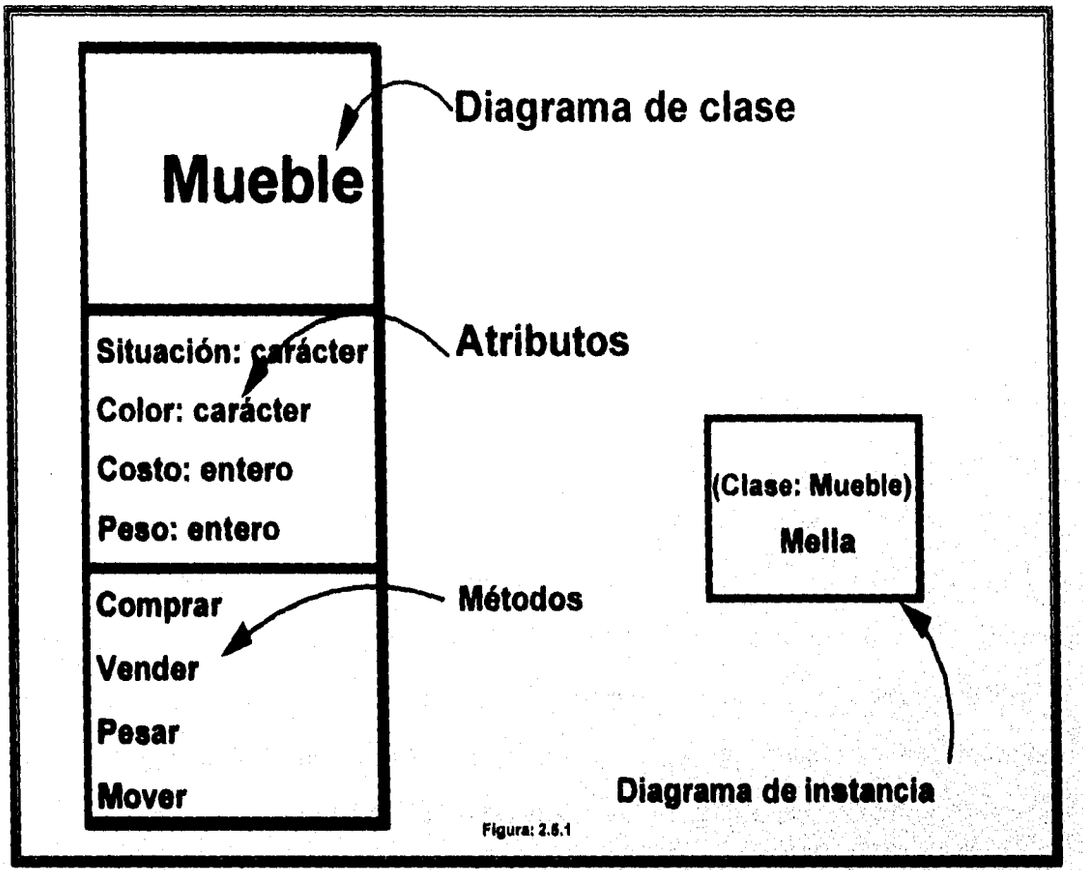


Figura: 2.6.1

Una vez que se ha definido la clase, se pueden reutilizar los atributos creando nuevas instancias de la clase. Por ejemplo, supongamos que vamos a definir un nuevo objeto denominado **mella** (un cruce entre una mesa y una silla) que sea miembro de la clase **mueble. Mella** hereda todos los atributos de **mueble**.

Cada objeto de la clase mueble pude ser manipulado de muchas formas, puede ser vendido y comprado, modificado físicamente (por ejemplo, se le puede quitar una pata o pintarlo de dorado), o movido de un sitio a otro. Cada una de esas operaciones (métodos) modifican uno o más atributos del objeto, por ejemplo una operación denominada **mover** puede modificar el atributo:

situación = edificio + piso + habitación.

Se puede utilizar el método **mover** para una mesa o una silla mientras que sean instancias de la clase **mueble.** Todos los métodos válidos (por ejemplo: **comprar, vender, pesar)** para la clase **mueble** están conectados a la definición del objeto tal

como muestra la siguiente figura y son heredadas por todas las instancias de la clase.

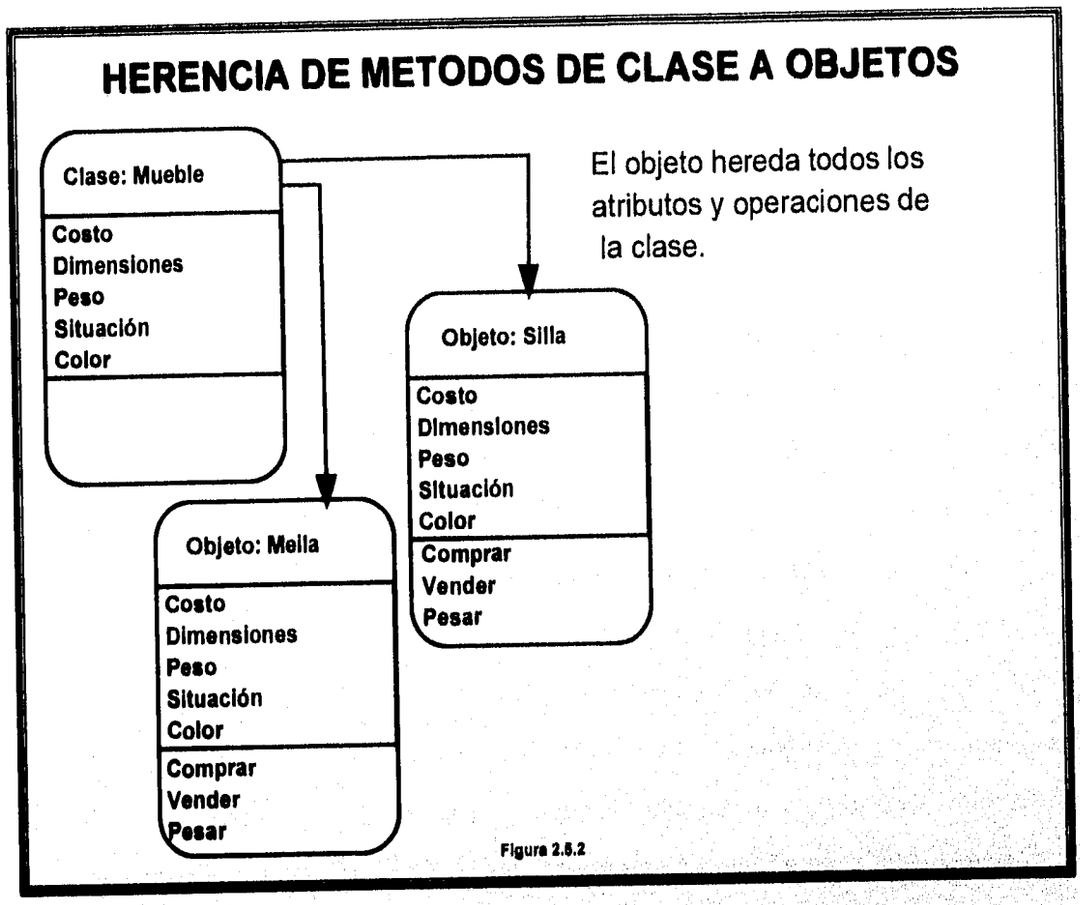


Figura 2.5.2

"Un término abstracto es como una valija de doble fondo: se puede poner en ella las ideas que uno desea y sacarlas de nuevo, sin ser observado."

Alexis de Tocqueville,
La democracia en América (1835)

2.6 ABSTRACCION

El término revolución industrial en el software se ha utilizado para describir el paso hacia una era en la que el software será compilado a partir de componentes de objetos reutilizables, con los cuales se crearía una enorme biblioteca de componentes. Debemos pasar de una era de productos de software monolíticos, donde un vendedor construye todo un complejo sistema hasta una era en que el software sea ensamblado a partir de componentes y productos de software de

muchos proveedores (de la misma forma que las computadoras y los automóviles son ensamblados a partir de componentes de muchos proveedores). Los componentes serán cada vez más complejos desde el punto de vista interno, pero será más sencillo interactuar con ellos, serán cajas negras donde no podremos mirar al interior. El diseñador de un programa no debe preocuparse muy temprano por la implementación concreta de sus estructuras de datos: arreglos, listas ligadas, etc. En vez de esto, se prefiere que el diseñador se concentre en entender qué cosas (pronto serán llamadas objetos) el programa maneja, y las características que tales cosas tienen desde el punto de vista de la aplicación. Por ejemplo, en el diseño de una base de datos orientada a objetos de renta de video-cintas a particulares (a familias, no a negocios), un "cliente" es un concepto importante que el sistema va a manejar; sus características son: nombre, domicilio, teléfono, qué cintas rentó, cuando las rentó, saldo, tipos principales de cintas que renta, y número total en cada tipo, en lo que va del año.

En cambio son de menor importancia (y pueden dejarse para el final, o incluso idealmente que el compilador o herramienta de programación se encargue de ellas) si un cliente es un arreglo o una pila. Comienzan sus elementos a contarse en 0 o en 1; el nombre del cliente cuantos caracteres tiene?. La orientación a objetos fomenta que los programadores y usuarios piensen sobre las aplicaciones en términos de objetos, los programadores buscan un factor de comportamiento común y lo sitúan en superclases abstractas. Las bibliotecas de clases proporcionan un depósito para los elementos comunes y reutilizables. La maquinaria de la herencia mantiene automáticamente las relaciones entre las clases dispuestas jerárquicamente en una biblioteca de clases. Los marcos estructurales contienen las bibliotecas de clases específicas de la aplicación. Cada nivel de abstracción facilita el trabajo de programación porque hay disponible una mayor cantidad de códigos reutilizables.

La modularización es una ventaja que nos brinda el paradigma de la OO.

Tipo de datos abstractos = Encapsulación + Ocultamiento de la información.

- La clase es un nivel cómodo y natural.

- Los módulos están centrados en los datos.

2.7 ENCAPSULACION

Es el término formal que describe el conjunto de métodos y datos dentro de un objeto, de forma que el acceso a los datos se permita solamente a través de los propios métodos del objeto. Ninguna otra parte de un programa orientado a objetos puede operar directamente sobre los datos de un objeto. La comunicación entre un conjunto de objetos sucede exclusivamente por medio de mensajes explícitos.

El encapsulamiento (ocultamiento de la información) es la separación de los aspectos externos de un objeto, que son accesibles a otros objetos, de los detalles de la construcción íntima del objeto, que son inaccesibles a los demás objetos.

La interfaz a través de la cual los demás objetos (clientes) se sirven del objeto (proveedor), es el subconjunto de las operaciones que son exportables.

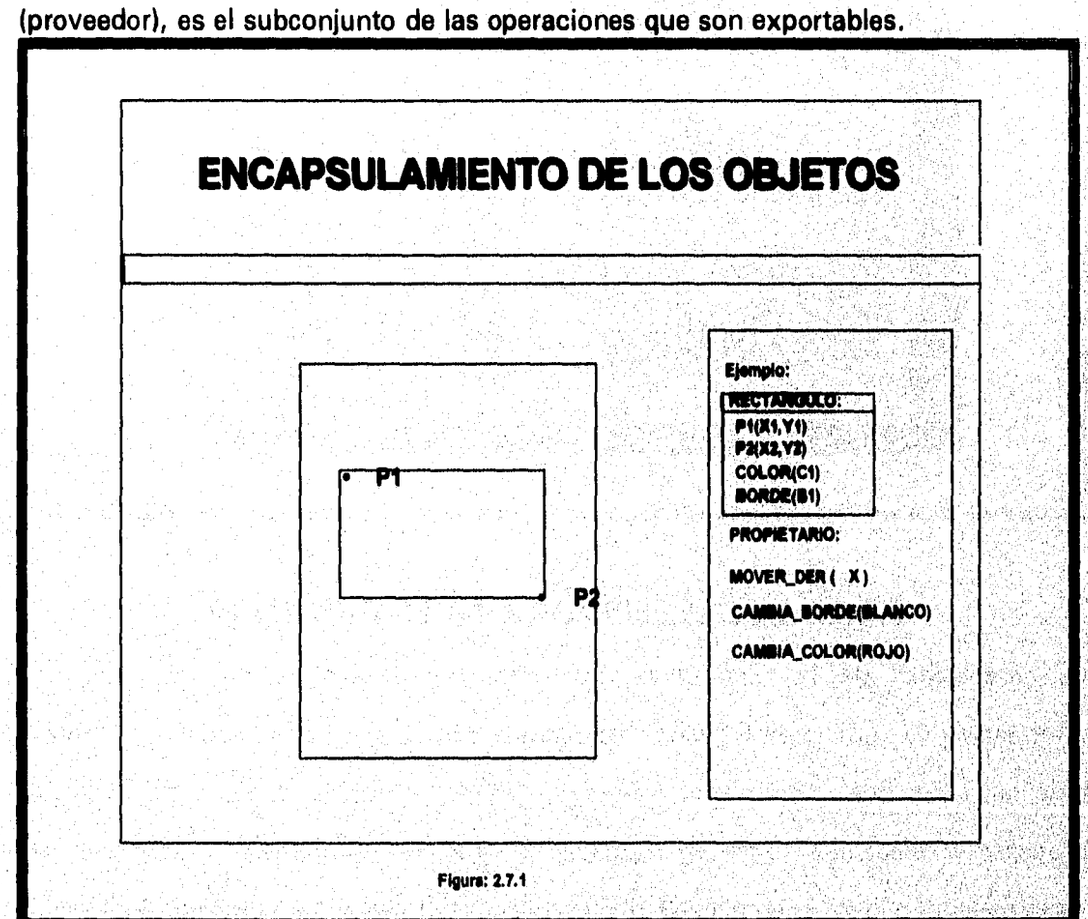


Figura: 2.7.1

En la figura 2.7.1, los datos que definen el rectángulo (P1, P2,C1,B1) quedan encapsulados en el interior del rectángulo, por lo que no son directamente accesibles ni modificables, sólo el objeto mismo los puede alterar a través de las operaciones en su repertorio.

De hecho los objetos tienen una representación interna en el computador, que por lo general tienen poco que ver con nuestra idea intuitiva, pero que facilitan su funcionamiento apegado a la realidad.

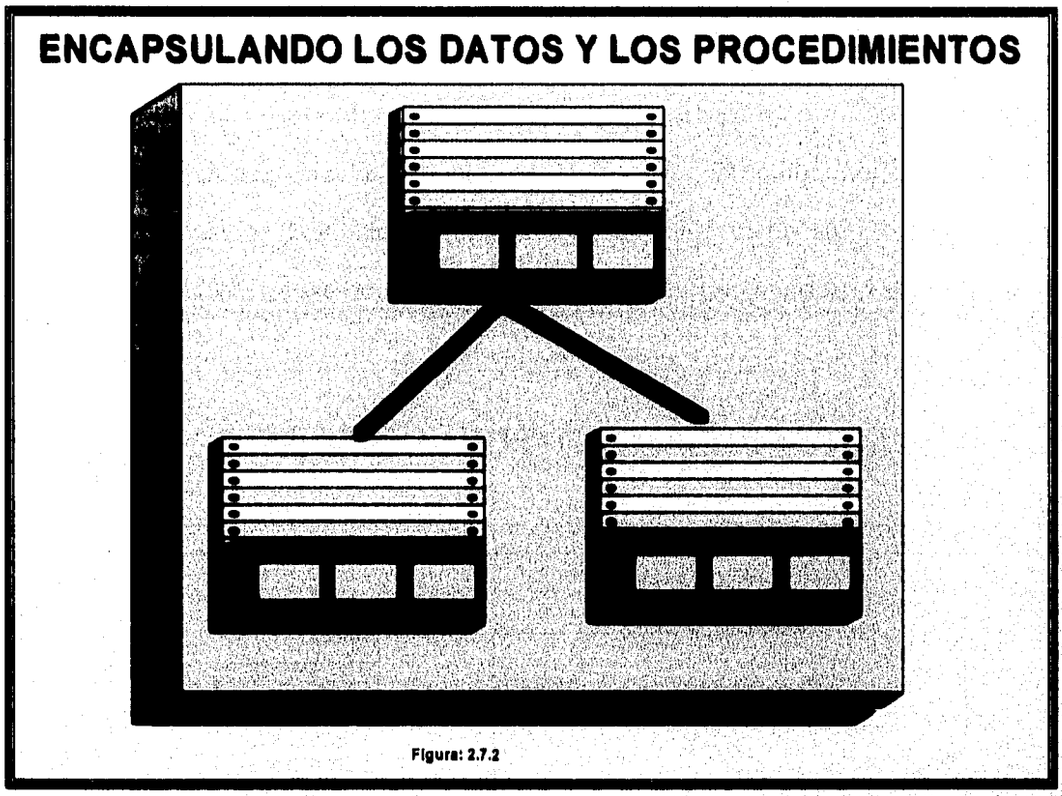


Figura: 2.7.2

2.8 LA COMPLEJIDAD DE LA ESTRUCTURA

La complejidad de las estructuras de datos va cambiando, los sistemas de información se apoyan en las bases de datos y éstas necesitan manejar estructuras más ricas de datos como son: Nombres, fechas, importes; también se necesita procesar y almacenar texto en forma libre, gráficas, diagramas, voz, y algunas secuencias de video, el empleo de documentos Multimedia que combinan texto con hojas de cálculo, gráficas, fotografías, y comentarios hablados. Necesitamos que las

BD soporten estructuras más ricas de datos, como ejemplo, la siguiente figura muestra un registro personalizado de Multimedia.

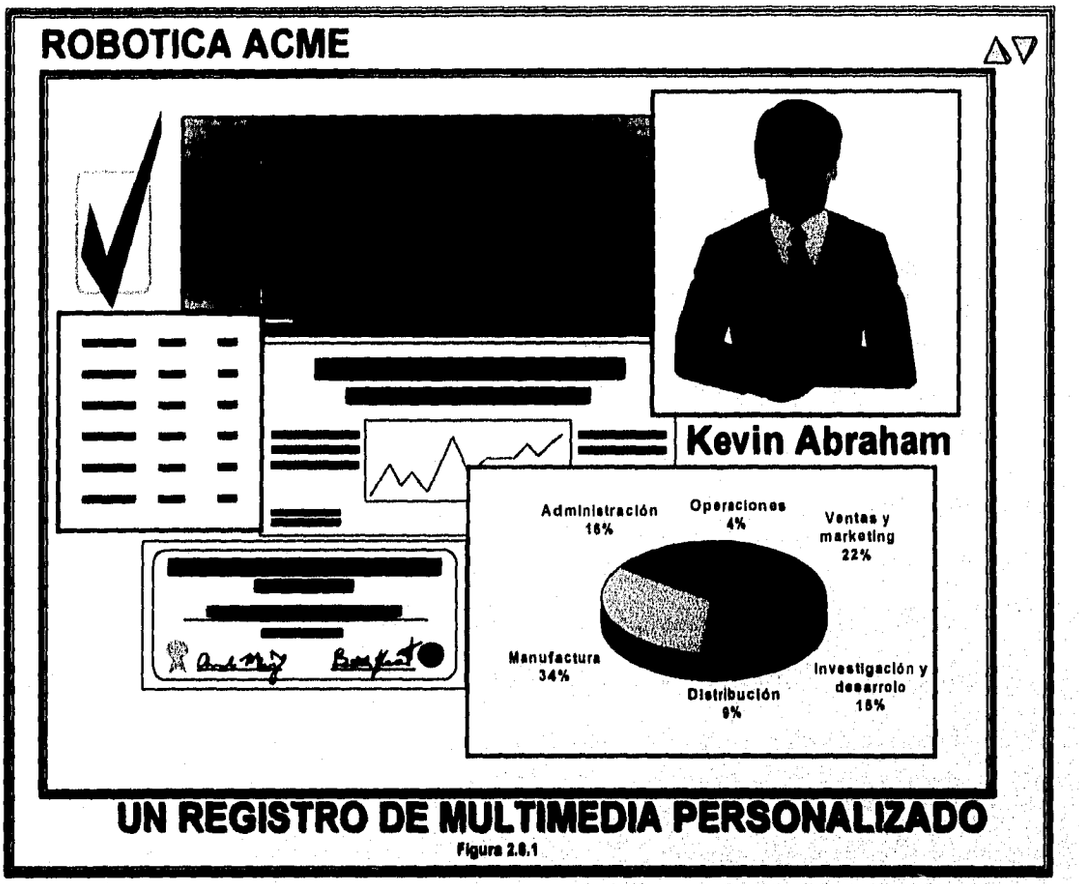


Figura 2.8.1

2.9 SOBRE CARGA DE OPERADORES

La OO permite que dos operaciones distintas puedan conocerse con el mismo nombre. Esto es común en los lenguajes de programación tradicionales, aunque en forma limitada, es decir, todos los lenguajes de programación utilizan el mismo símbolo para realizar operaciones aritméticas, independientemente del tipo de datos a operar. Por ejemplo: el símbolo + en a+b representa una suma, pero sus operandos pueden ser enteros, reales, complejos, etc., Pascal va más allá aún, donde el símbolo + en contextos apropiados también representa unión de conjuntos y concatenación de cadenas.

En la orientación a objetos el tipo de dato que se recibe determina el método correspondiente. Supongamos que tenemos un conjunto de rutinas para desplegar varios tipos de estructuras de datos, nos gustaría llamarlos a todos **desplegar,** pero sin la orientación a objetos no se puede, tendríamos que llamar desplegar_empleado, desplegar_empresa, etc. no es lo peor que nos puede pasar pero sí una molestia. En los lenguajes de programación orientados a objetos, el tipo de dato que está recibiendo un mensaje determina el método correspondiente; si mandamos el mensaje **desplegar** a un objeto empleado se activa el método para el despliegue de este tipo de dato, si mandamos este mismo mensaje a un objeto empresa es el método para desplegar empresas al que se invocará.

POLIMORFISMO

Polimorfismo es una palabra griega que significa muchas formas.

Varias clases pueden tener métodos definidos con el mismo nombre. Cuando se va a usar un método "genérico " o "sobrecargado" (que puede significar una de varias cosas), el objeto en particular nos dirá cual método especifico usar.

Un programa polimorfo contiene muchas funciones íntimamente relacionadas. Por ejemplo, en un sistema hay distintos tipos de datos, los cuales a su vez generan distintos tipos de reportes: recibos de nómina, facturas, listados de empleados, etc. El código para generar cada reporte es diferente, sin embargo la premisa principal para cada rutina es la misma "enviar algo a la impresora". Esta rutina tiene muchas formas, dependiendo de qué es lo que se va a imprimir.

2.10 LIGADURA DINAMICA

Ligadura dinámica es el proceso por medio del cual se proporciona al usuario de una rutina la dirección de ésta, en un lenguaje procedimental, la ligadura asocia al usuario de un procedimiento con la dirección del procedimiento. En un lenguaje orientado a objetos, la ligadura asocia un mensaje con el método para efectuar dicho mensaje. Esto puede suceder cuando el programa se compila y enlaza, o cuando se

está ejecutando. Si la ligadura ocurre durante el período de compilación-enlace, el proceso se denomina ligadura estática, ligadura temprana, ligadura fuerte o ligadura tiempo de compilación. Cuando la ligadura ocurre durante la ejecución, el proceso se denomina ligadura dinámica, ligadura tardía, ligadura débil, o ligadura de tiempo de ejecución. Los lenguajes tradicionales como C y Pascal soportan solamente la ligadura estática o ligadura tiempo de compilación. **La ligadura dinámica es una de las ventajas principales incluidas en los lenguajes orientados a objetos.**

Sin la ligadura dinámica, no es factible la flexibilidad y potencia que fomenta el paradigma, en el que los métodos son capaces de adaptarse automáticamente a los objetos a los que se aplican. No obstante la ligadura dinámica tiene un inconveniente: la reducción del rendimiento, ya que se necesita una búsqueda en tiempo de ejecución para hacer coincidir un método a cada mensaje.

Ejemplo: al ejecutar un programa orientado a objetos que invoca en un momento dado a un método llamado **dar_de_baja** definido para cada una de las subclases de la clase empleado, bajo este esquema será suficiente mandar el mensaje **dar_de_baja** al empleado "X" sin preocuparse por el subtipo (empleado_académico, empleado_administrativo, etc). Gracias a la ligadura dinámica, el lenguaje de programación que emplea esta técnica garantiza que el método que corresponde al subtipo presente se ejecutará, supongamos que se codifica el paso de un mensaje: Empleado_"X" -**dar_de_baja**.

Si Empleado_"X" fue declarado como empleado, ese es su tipo estático y significa que la variable puede asumir solamente valores de ese tipo, pero esto incluye también sus subtipos; al momento de ejecutarse el programa, el valor de empleado_"X" puede ser de cualquiera de los subtipos de la clase empleados, el ligado dinámico significa que el lenguaje de programación OO se encargará de determinar ese tipo dinámico y basado en él, invocará el método correspondiente al subtipo de empleado.

2.11 BLOB (BNARY LARGE OBJECT)

La reciente aceptación del procesamiento de imágenes aumenta la importancia de las bases de datos orientadas a objetos que puedan administrar los Blobs de manera eficiente. Actualmente el manejo de datos en un computador incluye multimedia (imagen, video, sonido, texto y números) y datos extensos con estructuras complejas que están representados en una gran cantidad de bits. BLOB es el acrónimo de "Binary Large Object", o sea Objeto Binario de gran tamaño, el término se refiere normalmente a un tipo de datos útil para almacenar grandes cantidades de datos. Además BLOB se utiliza para comparar bases de datos relacionales con bases de datos orientadas a objetos. En la mayoría de los casos (y en contraste con nuestra definición de objeto), la base de datos no interpreta el contenido de un BLOB (o sus métodos) sino más bien lo trata como una cantidad de datos sin diferenciar e insólitamente grande. **Un Blob es orientado a objetos, sólo si los métodos se encapsulan con los datos; existe capacidad de gestión de mensajes y el Blob es parte de una jerarquía de clases.**

Los blob's OO tienen métodos que permiten mostrarlos o utilizarlos con métodos de todo tipo, por ejemplo de seguridad, que permitan cifrarlos y descifrarlos. En el procesamiento de imágenes los Blobs se pueden presentar de manera rápida con baja resolución para presentarse después con alta resolución.

Enviaremos mensajes a los Blobs por medio de métodos para indicarles que se exhiban a sí mismos, se cifren a sí mismos, se liguen para su edición, etc.

Por ejemplo: **En una base de datos OO, una factura se podría mantener en una forma gráfica hecha a mano junto con las instrucciones orales del cliente, así como un registro alfanumérico. La imagen gráfica es un objeto; las instrucciones orales son otro objeto; el registro alfanumérico es otro y juntos forman un objeto único FACTURA.**

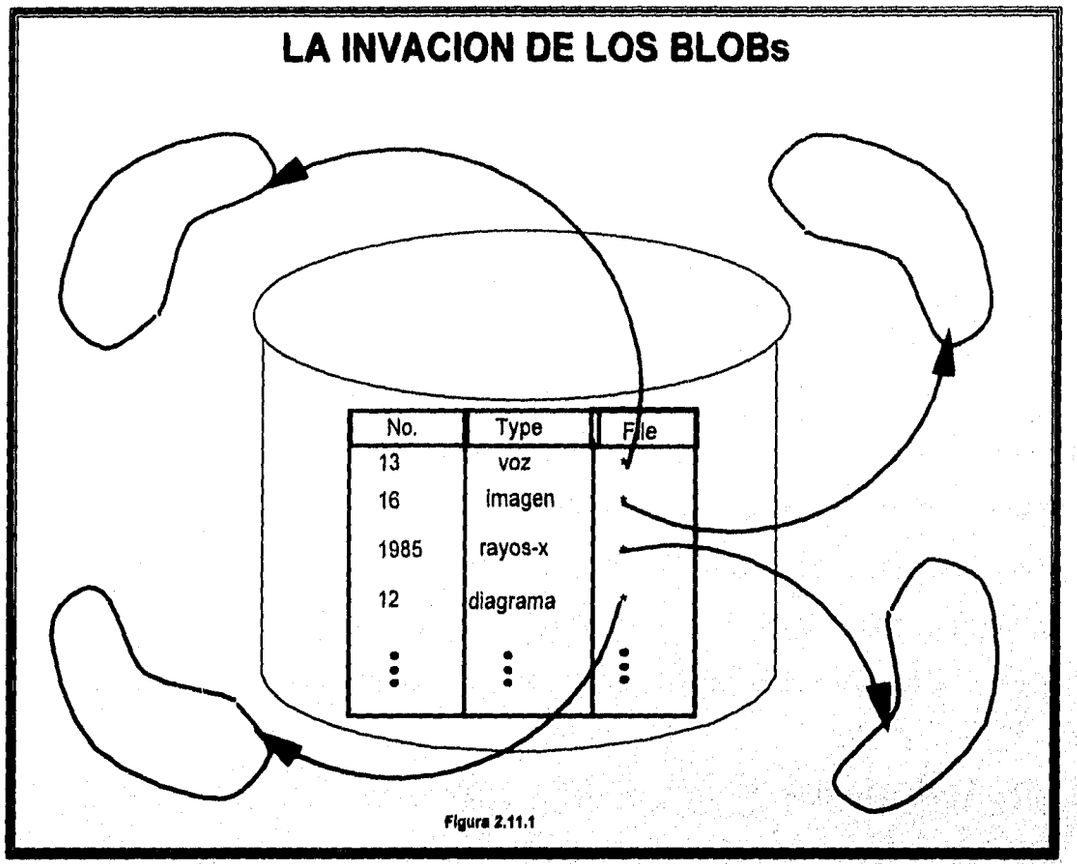


Figura 2.11.1

2.12 PROGRAMACION VISUAL

La programación visual se asocia a menudo con la orientación a objetos .

Un ejemplo de programación visual son los productos que recientemente han salido al mercado como ACCESS, VISUAL BASIC, DELPHI, FOX, PRO FOR WINDOWS, VISUAL OBJECT-CA, EXCEL, WORD, ETC.

La característica de estos productos visuales, es que las operaciones que tradicionalmente implicaban el llenar una tabla o escribir código, ahora con las herramientas visuales el código es generado en forma automática, y el llenado de tablas es actualizado por medio del puntero del ratón. Ejemplos: En Power Point de Micro Soft un objeto puede ser ampliado en pantalla no sólo cambiando los datos numéricos en una tabla dimensional sino manipulando la propia imagen, o en el caso de Access la creación de una llave primaria se desarrolla visualizando la tabla en forma de diseño, posicionandose en el campo que desee uno, que sea la llave

primaria y oprimir el botón de la barra de herramientas llave primaria. Las herramientas de windows, de copy y paste nos proporcionan un ejemplo sencillo de programación visual.

Aunque es cierto que algunas aplicaciones de programación visual proporcionan al usuario funcionalidad orientada a objetos, existen dos puntos concretos en los que no se cumple lo anterior:

Primero.- La programación visual no está intrínsecamente orientada a objetos. **Una imagen o ícono en pantalla no es un criterio suficiente ni necesario para considerar que una aplicación es orientada a objetos.**

Segundo.- Las aplicaciones de programación visual no están necesariamente construidas con lenguajes de programación orientados a objetos. De hecho, la mayoría de los productos visuales están construidos con lenguajes tradicionales.

2.13 LENGUAJES DE PROGRAMACION ORIENTADOS A OBJETOS LPOO

Hace años la programación se realizaba mediante el modelo lineal, es decir, los programas evolucionaban línea a línea sin que la descomposición modular de los mismos tuviesen la menor importancia a la hora de estructurar el código. Está situación fue superada por los métodos de la programación estructurada, donde la mera linealidad del código desaparecía para dejar paso a la noción de estructura modular, descomposición top-down, etc. Hoy vivimos otra nueva revolución en las técnicas de la ingeniería del software es lo que se ha dado ha llamar programación orientada a objetos. **Se dice que un lenguaje es basado en objetos si soporta objetos** como una de sus características; **se dice que es orientado a objetos si, adicionalmente, requiare que tales objetos pertenezcan a clases que puedan ser modificadas incrementalmente a través de la herencia (Wegner).** La expresión programación orientada a objetos se debe a Alan Kay.

Los lenguajes tradicionales no soportan la herencia, uno de los mecanismos más potentes de los lenguajes orientados a objetos, además la orientación a objetos

anima al desarrollador de sistemas a concentrarse en los temas importantes ignorando el resto. La orientación a objetos no es un concepto de nuevo acuño, de hecho tiene por lo menos 19 años de antigüedad, sus raíces pueden encontrarse en Noruega, a finales de los años 60's en conexión con un lenguaje llamado Simula 67, este lenguaje introdujo conceptos como son clases, corrutinas, subclases, muy parecidos a los lenguajes orientados a objeto de hoy en día. Posteriormente a mitad de la década de los 70's, los científicos del Centro de Investigación de Palo Alto de Xerox PARC crearon el lenguaje Smalltalk, cada elemento del lenguaje fue realizado como un objeto (Goldberg y Robson 1983). Con Smalltalk, cada aspecto del lenguaje, el entorno de programación y la cultura que lo rodeaba eran orientados a objetos, incluso hoy Smalltalk se considera el más puro de los lenguajes orientados a objetos.

Los entornos informáticos que surgieron en los años 80, y que hoy son típicos a mediados de los 90, son muy ricos gráficamente. En este entorno informático, las terminales inteligentes, o estaciones de trabajo proporcionan al usuario final una computación distribuida con acceso a múltiples bases de datos. Las aplicaciones soportan múltiples tipos de datos, incluyendo imagen, texto y video; llevados por esta complejidad, no es una sorpresa la aparición de los lenguajes, programación y bases de datos orientadas a objetos.

Dos grupos principales de los lenguajes orientados a objetos han surgido a partir de la última década de evolución del lenguajes orientado a objetos. Un grupo es el lenguaje orientado a objetos puro en el que casi todo es un objeto, en este grupo se encuentran Smalltalk, Actor, Iffel, y el otro grupo es el híbrido, es decir una mezcla entre los lenguajes tradicionales y los orientados a objetos, los miembros de este grupo incluyen a C++, Objective-C, Common Lisp Object System (CLOS), Pascal orientado a objetos, etc.

La siguiente figura muestra una parte de la biblioteca de clases de un lenguaje de programacion orientado a objetos:

PARTE DE LA BIBLIOTECA DE CLASES DE STEPSTONE PARA OBJETIVE-C

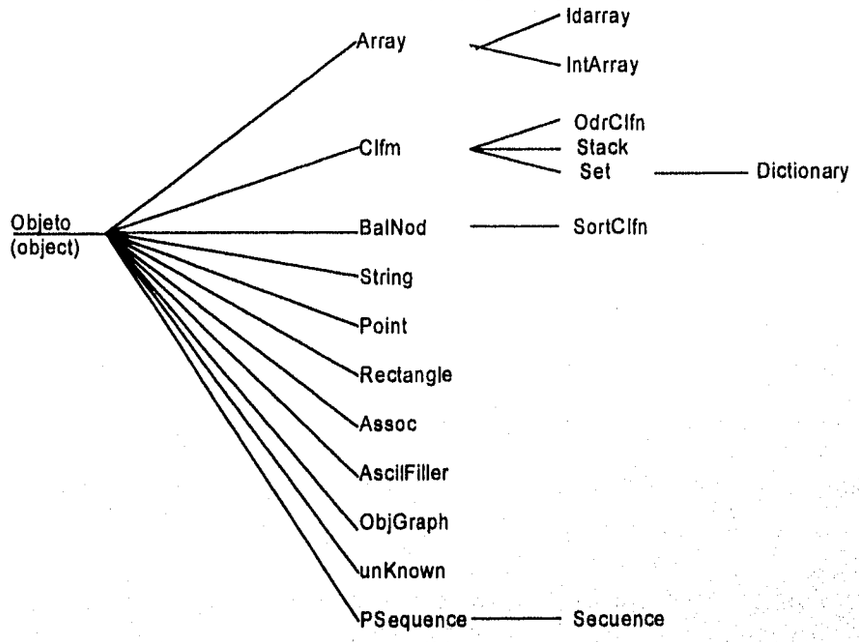


Figura 2.13.1

En general, los IPOO puros resaltan la exploración y el prototipo rápido, mientras que los híbridos ponen énfasis en la velocidad de ejecución y facilitan la incorporación del programador orientado a procedimientos, incorporando extensiones orientadas a objetos a los lenguajes procedimentales.

Un lenguaje de programación orientado a objetos beneficia al desarrollador de software proporcionándole una forma natural de modelar el fenómeno del complejo mundo real, los LPOO tienen el inconveniente de que su ejecución es más lenta que los lenguajes proceduales, debido a que como ya se vió anteriormente, el concepto de ligadura dinámica implica la compilación al momento de ejecución, según Deutsch 1989, Smalltalk se ejecuta cinco veces más lento que C.

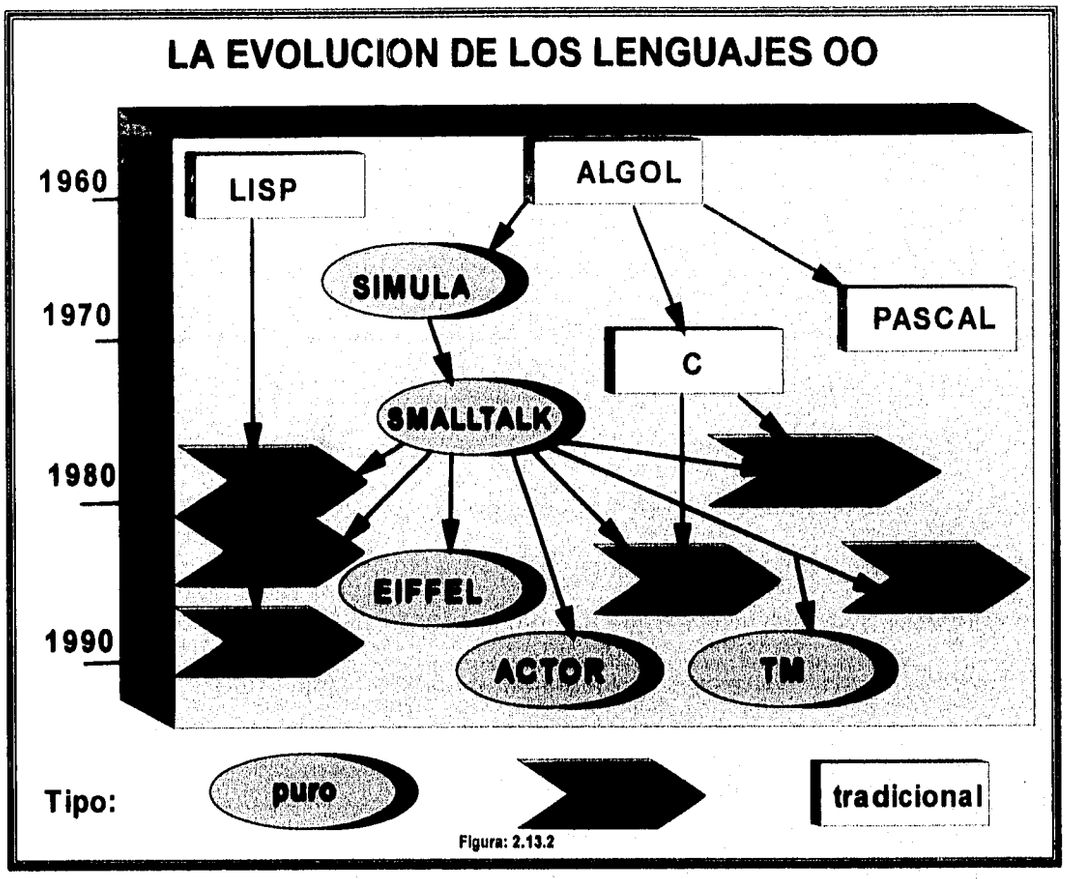


Figura: 2.13.2

Un programa OO = universo de objetos comunicándose por medio de envío de mensajes, en donde cada objeto es responsable de sí mismo.

[illegible]	[illegible]	[illegible]
Ejemplo	Fortran, Cobol, Algol, Pascal, C	Simula, Smalltalk, C++, Ifefel,TM
	jerarquía de procedimientos	organización dinámica, más plana
	procedimientos ejecutan una tarea	objetos se comunican entre sí
imperativo	procedimiento = comando	operación básica = comando ejecución = secuencia de acciones
	procedimientos manipulan datos	métodos que responden a mensajes
	todo poderoso para interpretar comandos	cada objeto es responsable de sí mismo. (no se ve un todo poderoso)

Las tendencias en la tecnología de lenguajes de programación OO

La siguiente generación de lenguajes será consecuencia de la fusión de:

Medio ambiente C

Medio ambiente C + +

UNIX MACH.

Bases de Datos Orientadas a Objetos.

Bases de Datos Relacionales extendido a Objetos.

Tendrán apariencia de Windows.

Los lenguajes de programación estarán íntimamente relacionados con las BDOO.

Estarán distribuidos en una red local.

La programación no se hará en líneas de código sino en forma interactivo - visual.

Se fusionarán también lenguajes de programación lógica, estilo Prolog.

2.14 UN MODELO CONCEPTUAL UNIFICADO

En la computación tradicional, los modelos conceptuales para cada una de las etapas del ciclo de vida de un sistema de bases de datos (análisis, diseño, definición y acceso a la BD) son diferentes. Por el contrario, las técnicas de OO emplean el mismo modelo conceptual para cualquier etapa del ciclo de vida de un sistema incluyendo la definición y acceso a la base de datos. La tecnología de las Bases de datos OO dá un paso más hacia la unificación.

Ciclos de vida de los sistemas OO

Cuando desarrollamos un sistema OO siguiendo el método de la cascada OO, la reducción del tiempo de desarrollo comparando con los modelos tradicionales (codifica y fija, cascada, evolutivo, transformativo, espiral, etc.) es hasta de un 50% menor que con el paradigma tradicional.

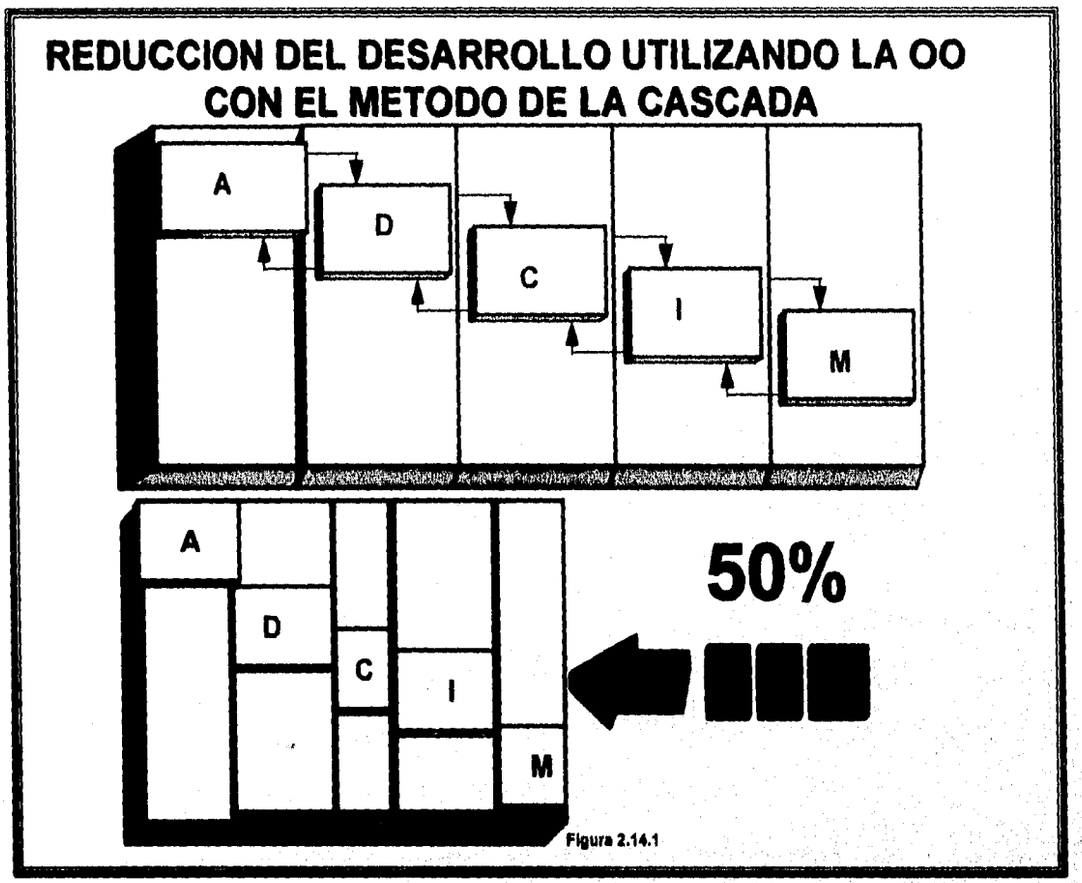


Figura 2.14.1

Modelos OO.

Cuando analizamos sistemas, creamos modelos del área de aplicación que nos interesa. Un modelo puede incorporar un sistema, centrarse en una área de la empresa o abarcar toda la empresa. El modelo representa un aspecto de la realidad y se construye de modo que nos ayude a comprender a ésta. El modelo es mucho más sencillo que la realidad, al igual que un porche a escala es más sencillo que uno de verdad, podemos manejar modelos y ésto nos ayuda a idear sistemas o rediseñar áreas de la empresa. **La forma de modelar la realidad con el análisis orientado a objetos, difiere del análisis convencional ya que con el análisis OO modelamos al mundo en términos de tipos de objetos y lo que le ocurre a éstos, después de todo la realidad del mundo esta compuesta de objetos y eventos que cambian el estado de dichos objetos.**

El análisis de requerimientos basado en objetos esencialmente consiste en observar.

Un método de análisis y diseño orientado a objetos debe proponer los pasos a seguir para construir el modelo del sistema en términos de clases y objetos relaciones, entre ellos, esta receta se conoce como el proceso de análisis y diseño. Durante el proceso de análisis se busca definir las clases semánticas del dominio del problema y luego durante el diseño, se trata de extenderlo hacia el dominio de la solución.

Los modelos se expresan a través de alguna notación. En la ingeniería del software, los modelos generalmente se expresan en forma gráfica o textual.

Nivel Conceptual	modelos
Nivel lingüístico	diagramas, notaciones
Nivel factual	objetos

Como las expresiones gráficas y textuales posen un lenguaje subyacente, los modelos incluyen una semántica y una pragmática; y la modelación es la técnica más poderosa para reducir la complejidad.

La modelación del mundo con objetos comprende la elaboración de tres modelos complementarios :

1. **El modelo estático** (modelo de los objetos o modelo de la información, **especifica a quién le sucede**) **Este modelo captura los objetos en el sistema, las relaciones existentes entre ellos y los atributos y operaciones que caracterizan a cada clase de objetos. Como su nombre lo indica, la modelación orientada a objetos enfatiza los objetos del sistema más que su funcionalidad, como sucede en la modelacion tradicional orientada a funciones.**
2. **El modelo dinámico** (modelo de estados, **especifica cuando sucede**). **Una vez que se ha elaborado una primera versión del modelo estático de la composición y la estructura de un sistema, es preciso examinar los cambios en relación con el tiempo. Los aspectos de un sistema que tienen que ver con el tiempo y la**

mutabilidad son el modelo dinámico. Además otro aspecto importante de un sistema OO es el control, que describe las secuencias de operaciones que ocurren como respuestas a los estímulos externos sin considerar qué operaciones realizan, qué es lo que transforman o como se implementan. Los principales conceptos de la modelación dinámica son los eventos (estímulos externos) y los estados (valores que los objetos pueden tomar). Los diagramas de estados son un concepto matemático usual (representaciones gráficas de máquinas de estados) que se manejan en la literatura desde diferentes perspectivas. La notación empleada es la desarrollada por David Harel.

3. **El modelo Funcional** (modelo de los procesos, **especifica que sucede**). El modelo funcional describe las computaciones que el sistema habrá de realizar. De este modo, el modelo funcional muestra cómo se derivan los valores de salida en una computación sin considerar cuándo y cómo se computan los valores. El modelo funcional especifica también el significado de las operaciones y las restricciones del modelo estático, así como las acciones del modelo dinámico. Este modelo consiste de múltiples diagramas de flujo de datos, (diagramas de burbujas) que muestran las relaciones funcionales entre los valores computados por el sistema, incluyendo valores de entrada, valores de salida y almacenes internos de datos.

El análisis en el caso del método de Booch, no se presenta como una secuencia de pasos a seguir sino más bien, se plantea como una serie de preguntas que debe contestar el diseñador para realizar la construcción del modelo. Las preguntas son las siguientes:

- **¿ Qué clases conforman el sistema y cómo se relacionan entre sí ?**
- **¿ Cómo están estructurados los objetos individuales de estas clases y cómo colaboran entre sí?**
- **¿ Dónde estarán definidas las clases y creados sus objetos ?**
- **¿ A qué procesador se asociarán los objetos activos y cómo se organizará el manejo de hilos de control, la comunicación y la sincronización en el caso de los sistemas concurrentes y/o distribuidos ?**

Las primeras dos preguntas están relacionadas con la estructura lógica global del sistema, mientras que las dos últimas se refieren a las decisiones de diseño que hay que tomar con respecto al mapeo físico del sistema, a los módulos de programas y a la arquitectura particular de la(s) máquina(s).

Construir el modelo del sistema consiste en ir contestando estas preguntas en cualquier orden, claro que al principio uno empieza por las dos primeras, pero a medida que se avanza con las respuestas se pueden ir tomando decisiones que implique modificaciones de cualquier nivel, afectando ya sea la estructura lógica o física del sistema.

El proceso de análisis consiste principalmente en ir descubriendo las clases de objetos que modelan el dominio del problema, mientras que el diseño requiere de más invención de clases adicionales y de adaptación de lo previamente modelado. El proceso de análisis y diseño es evolutivo.

El primer paso del análisis es descubrir en la descripción del problema los candidatos para las clases y los candidatos para los métodos. Boch sugiere que los candidatos para clases abstractas se busquen en sustantivos significativos en la descripción, mientras que los candidatos para los métodos deben buscarse en los verbos. Es decir, las clases del modelo de objetos son abstracciones que representan a objetos ya sean concretos o conceptuales, y los métodos mapean las acciones que estos objetos realizan.

Analisis OO

Es la liga entre los usuarios y los diseñadores. Debe proporcionar una descripción completa del problema, legible, revisable por las partes interesadas y verificable contra la realidad, para ello el analista genera un modelo en base a la identificación de clases y objetos que forman el vocabulario del dominio del problema.

Origen de las clases:

Cosas tangibles (autos, sensores,...)

Las primeras dos preguntas están relacionadas con la estructura lógica global del sistema, mientras que las dos últimas se refieren a las decisiones de diseño que hay que tomar con respecto al mapeo físico del sistema, a los módulos de programas y a la arquitectura particular de la(s) máquina(s).

Construir el modelo del sistema consiste en ir contestando estas preguntas en cualquier orden, claro que al principio uno empieza por las dos primeras, pero a medida que se avanza con las respuestas se pueden ir tomando decisiones que implique modificaciones de cualquier nivel, afectando ya sea la estructura lógica o física del sistema.

El proceso de análisis consiste principalmente en ir descubriendo las clases de objetos que modelan el dominio del problema, mientras que el diseño requiere de más invención de clases adicionales y de adaptación de lo previamente modelado. El proceso de análisis y diseño es evolutivo.

El primer paso del análisis es descubrir en la descripción del problema los candidatos para las clases y los candidatos para los métodos. Boch sugiere que los candidatos para clases abstractas se busquen en sustantivos significativos en la descripción, mientras que los candidatos para los métodos deben buscarse en los verbos. Es decir, las clases del modelo de objetos son abstracciones que representan a objetos ya sean concretos o conceptuales, y los métodos mapean las acciones que estos objetos realizan.

Analisis OO

Es la liga entre los usuarios y los diseñadores. Debe proporcionar una descripción completa del problema, legible, revisable por las partes interesadas y verificable contra la realidad, para ello el analista genera un modelo en base a la identificación de clases y objetos que forman el vocabulario del dominio del problema.

Origen de las clases:

Cosas tangibles (autos, sensores,...)

Papeles que representan (madre, profesor,...)

Eventos que ocurren (aterrizaje, petición,...)

Interacción (préstamo, encuentro,...)

Organizaciones (Escuela, ONU,...)

Conceptos (negociación, comunicación,...)

Nombres de las abstracciones:

Objeto sustantivo (el_sensor, un_sólido,...)

Clase nombre común (sensor,sólido,...)

Operaciones que modifican verbos activos (dibuja, mueve,...)

Operaciones de selección preguntas (esta_abierto,...)

Diseño OO

Puede comenzar cuando la etapa de análisis no necesariamente ha concluido. Se sugiere analizar un poco y luego diseñar un poco. Concluye al establecer las abstracciones claves y mecanismos importantes, dejando para la implantación los aspectos de diseño que tienen poca o ninguna relación con la conducta observable del sistema; cuando las abstracciones claves son tan simples que no requieren mayor descomposición y pueden elaborarse con componentes reutilizables existentes.

Se crean abstracciones y mecanismos que proporcionan el comportamiento querido por el modelo.

Notación de Diseño:

La notación es un medio para documentar el diseño de un sistema.

Características deseables:

- Definición precisa.
- Expresiva.
- Estándar.

- Independiente del lenguaje de programación.

Instrumentación (Evolución) de un Sistema OO

Comprende los aspectos tradicionales de la programación, verificación e integración. Consiste de una producción incremental de series de prototipos, que evolucionan a la implantación final. Los posibles cambios de diseño comprenden:

- Añadir una clase.
- Cambiar la implantación de una clase.
- Cambiar la representación de una clase.
- Reorganizar la estructura de clase.
- Cambiar la interfaz de una clase.

Mantenimiento de un Sistema OO

Un programa que se usa en un ambiente real necesariamente debe cambiar. Los cambios que se le practican comprenden la introducción de nuevas funcionalidades que no estaban previstas en el problema original.

Administracion de Proyectos OO

En la asignación de recursos el Diseño OO, comparado con el diseño tradicional incurre en:

- Gastos equivalentes en análisis y mayores en diseño.
- Gastos mucho menores en programación y verificación.
- Considerablemente menores en integración (Incremental).
- Recursos humanos equivalentes o menores.
- Productos de mayor calidad.

Beneficios del Diseño OO

- Utiliza el poder expresivo de los lenguajes de programación OO.
- Favorece la reutilización del Software.

- Elabora sistemas flexibles al cambio.
- Reduce riesgos de desarrollo.
- Es atractivo a la cognición humana.

Riesgos del Diseño OO

- Desempeño menor, con respecto a un sistema diseñado tradicionalmente (poco maduro).
- Costos iniciales de adopción del método.

2.15 RELACION COSTO / BENEFICIO DE LA TECNOLOGIA DE OO

PRINCIPALES BENEFICIOS:

Las principales ventajas de la orientación a objetos radican en poder hacer frente a dos temas esenciales de la Ingeniería del Software que son Gestión de la Complejidad y Mejora de la Productividad en el Proceso de Desarrollo del Software. La orientación a objetos conduce estos temas fomentando las siguientes estrategias del desarrollo de software:

- **Reutilización.**- Las clases están diseñadas para que se reutilicen en muchos sistemas; para maximizar la reutilización, las clases se construyen de tal manera que sea fácil poderlas adaptar, lo que permite que el desarrollo del software sea en fracción de tiempo y costo respecto a los métodos tradicionales. Un objetivo fundamental de las técnicas OO es lograr la reutilización masiva en la construcción del software.
- **Estabilidad.**- Las clases diseñadas para una reutilización repetida se vuelven estables, en forma análoga que los microprocesadores se hacen estables.
- **Abstracción superior.**- El diseñador piensa en cómo se comportan los objetos y no en detalles de bajo nivel, el encapsulado oculta los detalles y hace que las clases complejas sean fáciles de utilizar. Otro objetivo de las técnicas OO es lograr el mayor grado de abstracción.

- **Software complejo pero confiable.** - Se construyen clases a partir de otras clases, las cuales a su vez se integran mediante clases, formando clases muy complejas pero confiables ya que existe una gran probabilidad de que el software construido a partir de clases estables, probadas y perfeccionadas tenga menos fallas que el software construido a partir de cero.
- **Apertura del mercado del software.** - La era del software artesanal o monolítico termina cuando se inicia una nueva era en donde los proveedores de software se especializan cada vez más en bibliotecas especializadas que se adapten con facilidad a las aplicaciones desarrolladas por los usuarios.
- **Diseño rápido y de calidad.** - Los diseños suelen tener mayor calidad, puesto que se integran a partir de componentes probados, que han sido depurados y perfeccionados varias veces y por lo tanto son más rápidos ya que se crean a partir de dichos componentes, que además son lo suficientemente portables y flexibles como para poderse adaptar a la plataforma y /o ambiente necesario.
- **Integridad.** - La protección contra la corrupción de información es una de las características de la tecnología de orientación a objetos, pues al tener encapsulados los datos con los métodos (procedimientos) que manipulan dichos datos no permite que otros métodos "externos " accesen dichos datos, lo cual es una ventaja en sistemas cliente/servidor o sistemas distribuidos al no permitir que usuarios no autorizados accesen al sistema a no ser por medio de los métodos de la clase de dicho objeto.
- **Modelado de sistemas más realista.** - El modelo OO modela a la empresa o área de aplicación de una forma más real, de lo que se logra con los modelos convencionales ya que en estos el modelo difiere cada vez que cambiamos de una fase del ciclo de vida a otra (del análisis al diseño y/o del siseño a la implantación), con las técnicas OO el modelo se unifica, lo cual además permite que los usuarios (empresarios, etc.) compartan con el desarrollador un modelo "inteligente[@] " común, ya que el empresario piensa en términos de objetos,

[@] Los modelos empresariales describen las reglas que controlan a una empresa, estas se expresan en terminos de eventos, además la forma en que estos deben modificar el estado de los objetos en la empresa.

eventos y políticas empresariales que describen el comportamiento de los objetos.

- **Independencia del diseño.** - Las clases están diseñadas para ser totalmente independientes del ambiente de la plataforma de hardware o de software, ya que los mensajes utilizan solicitudes y respuestas con formato estándar. Esto les permite ser utilizadas en diferentes sistemas operativos, controladores de bases de datos, controladores de redes, etc. El creador de software no tiene que esperar a que se le especifique el ambiente.
- **Computación paralela.** - La rapidez de las computadoras mejorará mediante la construcción de computadoras de procesamiento paralelo, el multiprocesamiento se efectuará "al mismo tiempo" en diferentes chip's, (con el tiempo un chip tendrá varios procesadores). Los objetos de procesadores distintos se ejecutarán de manera simultánea, actuando cada no en forma independiente. Un object Request Broker estándar permitirá que las clases de procesadores independientes envíen respuestas a otras.
- **Mayor nivel de automatización de las bases de datos.** - Las estructuras de datos en las bases de datos OO están ligadas a métodos que realizan acciones automáticas. Una base de datos OO tiene integrada una "**inteligencia,**" en forma de métodos, en tanto que una base de datos relacional básica no.
- **Eficacia de la máquina.**- Ha quedado demostrado que las bases de datos orientadas a objetos tienen un rendimiento mucho mejor que las bases de datos relacionales, para ciertas aplicaciones con estructuras de datos muy complejas. Esto, aunado al multiprocesamiento con el diseño OO, promete un salto enorme en el desempeño de las máquinas. Los sistemas cliente /servidor basados en Lan utilizarán máquinas despachadoras con concurrencia y **bases de datos orientadas a objetos.**

PRINCIPALES COSTOS:

- **Migración.** - Dado que la adopción de esta metodología implica un cambio radical, las desventajas de la tecnología salen a flote cuando el usuario trate de migrar de una plataforma que no está basada en los objetos, a un sistema orientado a objetos. La mayoría de los proveedores dicen que la aceptación de las aplicaciones administrativas heredadas se resolverá "envolviendo" la aplicación antigua en un armazón orientado a objetos que la haga aparecer al administrador basado en objetos como otro objeto, de modo que la comunicación con ellas sea a través de mensajes estándar OO. Pero esto es un problema, ya que la inversión actual de las empresas esta basado en sistemas relacionales que han costado demasiado. Como toda tecnología, la de la orientación a objetos requiere de una cuidadosa planeación del proceso de migración.
- **Adiestramiento del personal.**- El Adiestramiento del personal de sistemas en desarrollo, con la aplicación de estas nuevas herramientas deberá ser tan importante, que aunado a investigaciones de nuevas y diferentes plataformas de cómputo y software, soporten este nuevo enfoque de desarrollo. El costo bien vale el beneficio, ya que si invertimos en la capacitación del personal para que desarrolle bajo esta tecnología, el beneficio será mayor.

CAPITULO III

BASES DE DATOS ORIENTADAS A OBJETOS

> Creo que hasta 1996-1997 las realizaciones de bases de datos relacionales habrán alcanzado su punto máximo. El siguiente paso en su evolución serán las bases de datos orientadas a objetos y el procesamiento de información basado en el conocimiento y fundamentado en el modelo semántico. Las bases de datos tienen que representar la vida real, y ésta es complicada.
>
> Shaku Atre, citado en Chorafas (1989).

CAPITULO III

BASES DE DATOS ORIENTADAS A OBJETOS

3.1 LA EVOLUCION DE LOS DBMSOO

Dentro de la tecnología de punta de la OO se encuentran las Bases de Datos Orientadas a Objetos (BDOO), tambien llamadas bases de objetos, las cuales han evolucionado como resultado de las actuales limitaciones del modelo relacional en cuanto a su incapacidad de soportar numerosos tipos de datos complejos, las BDOO han tomado lo mejor de sus predecesoras para formar así, una mejor base de datos. Los sistemas administradores de BDOO (DBMSOO) estan madurando en forma vertiginosa y se han introducido poco a poco en el mercado comercial. Los DBMSOO son básicamente similares a las generaciones previas de DBMSs ya que contienen mecanismos para almacenar y recuperar información, manejar accesos concurrentes, seguridad de información, respaldo y restauración de información, y la ejecución de los servicios que los DBMS tradicionales brindan. Lo que los hace diferentes es que almacenan objetos, con un mejor manejo de las estructuras de datos, un tanto mejor que los DBMS jerárquicos, networks o Relacionales. En forma similar los DBMSOO pueden ser vistos como un paso más en la evolución tecnológica de estructuras de información de los DBMS. Las implicaciones van más lejos que ésto porque los objetos contienen no sólo datos sino también sus métodos (procedimientos). Este es un importante cambio que rompe los roles tradicionales de un DBMS, esto tiene efectos significativos en el papel de los DBMS y en incorporar sistemas de información. Un sistema Administrador de Bases de Datos Orientado a Objetos es un sistema de base de datos (DBMS) que ofrece un modelo de datos orientado a objetos. Estos DBMSOO nacieron al inicio de los 80, sin embargo, tenían varias características que limitaban su empleo en el área comercial. En primer lugar la orientación hacia el diseño suponía que el usuario ejecutaba un número limitado de transacciones ampliadas, en comparación con las transacciones

frecuentes y de gran volúmen de muchas aplicaciones comerciales. En segundo lugar, los usarios de ingeniería al igual que los programadores, accedían a los objetos a través de un browsing mientras que los usuarios comerciales requerían utilidades de consulta fáciles de utilizar como SQL, además los vendedores de DBMSOO proponían alternativas al SQL sin tener éxito, ya que SQL se ha convertido desde hace tiempo en un estandar del área relacional; finalmente, estos primeros DBMSOO adolecían de bajo rendimiento, haciendolas inadecuadas para aplicaciones de gran escala y de gran volúmen. Productos nuevos y mejoras en las versiones iniciales han corregido muchas de estas limitaciones, evolucionado hasta nuestros días, desde entonces ofreciendo las ventajas de la orientación a objetos además de las que comunmente ofrecen los tradicionales DBMS. **Una base de datos Orientada a objetos almacena, modifica y recupera Objetos por orden de un programa de aplicación, dichos objetos pueden ser tan simples como una cadena de caracteres, números o bien tan complejas como las especificaciones completas de una tarjeta de circuitos, o una aplicación multimedia (sonido, gráficos y video).** Una de las principales ventajas que tiene una BDOO sobre sus predecesoras es la capacidad de poder representar como objeto cualquier entidad del mundo real.

El siguiente ejemplo muestra cómo una base de datos sobre datos académicos es representada en tablas mediante el modelo relacional

Se observa cómo un objeto puede contener otros objetos en cualquier nivel de anidamiento, proporcionando así una gran flexibilidad al definir nuevos tipos de objetos. Como se muestra en el ejemplo, los objetos hacen más fácil el desarrollo de aplicaciones, debido a que todos los datos de una entidad están localizados en un sólo lugar. El desarrollador no necesita buscar a través de múltiples tablas relacionadas por medio de apuntadores para determinar donde está almacenada

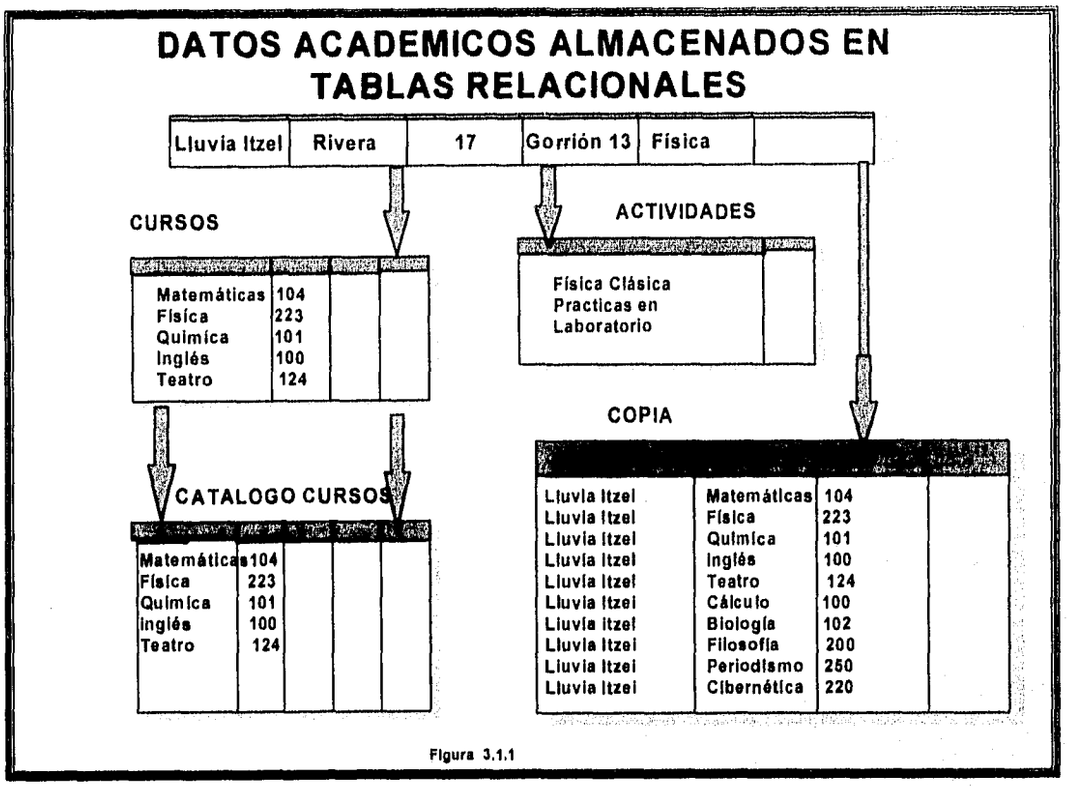


Figura 3.1.1

Y a su vez podemos representar la misma base de datos como un conjunto de objetos relacionados.

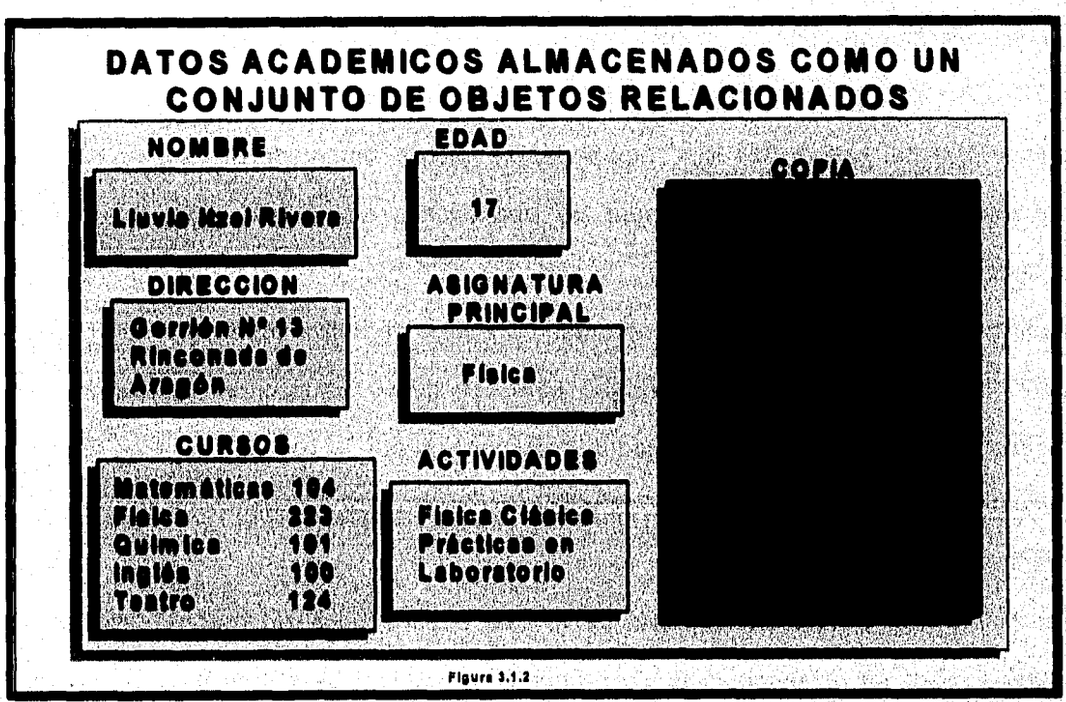


Figura 3.1.2

por ejemplo la copia del estudiante. Además de los datos los objetos pueden almacenar relaciones, representadas internamente como enlaces a otros objetos y pueden almacenar el comportamiento representado internamente como métodos.

Los objetos representan además, entidades u objetos del mundo real que esta siendo modelado en la aplicación, objetos en el sentido de combinaciones encapsuladas de estructuras de datos (atributos y propiedades) y procedimientos asociados (métodos) que describen su comportamiento. **El mapeo uno a uno reduce la distancia semántica entre el mundo real y el modelo utilizado para representarlo dentro de la computadora. Más aún, asociado a un estilo de programación OO el DBMSOO reduce la diferencia semántica entre el programa y la Base de Datos que lo soporta.**

Los objetos hacen también que las aplicaciones se ejecuten más rápido. Como los datos de una entidad están relacionados lógicamente, la BDOO tiene los medios para optimizar su localización física, las aplicaciones son capaces de leer menos archivos para recuperar todos los datos relevantes.

Los sistemas tradicionales de CAD, por ejemplo, almacenan a menudo cada componente de un diseño en un archivo separado y utilizan una base de datos para almacenar los nombres de los archivos. Para acceder al diseño compílo, la aplicación CAD debe abrir y cerrar todos los archivos de la lista de archivos. En una BDOO, el diseño entero con todos sus componentes puede almacenarse en un objeto, reduciendo grandemente el número de operaciones de archivo necesarias para acceder al diseño.

A diferencia de las BDR que sólo permiten conjuntos de datos del mismo tipo (char, date, long, double, etc.), las BDOO permiten conjuntos arbitrarios de objetos, dichos conjuntos de objetos pueden manipularse por el usuario o bien por la aplicación, bloqueados para gestión de transacciones, o agrupados para optimizar el rendimiento y facilidad de acceso.

No existe una definición estandar de lo que es una base de datos orientada a objetos, pero se ha establecido un primer concenso en la definición y conceptos

por ejemplo la copia del estudiante. Además de los datos los objetos pueden almacenar relaciones, representadas internamente como enlaces a otros objetos y pueden almacenar el comportamiento representado internamente como métodos.

Los objetos representan además, entidades u objetos del mundo real que esta siendo modelado en la aplicación, objetos en el sentido de combinaciones encapsuladas de estructuras de datos (atributos y propiedades) y procedimientos asociados (métodos) que describen su comportamiento. **El mapeo uno a uno reduce la distancia semántica entre el mundo real y el modelo utilizado para representarlo dentro de la computadora. Más aún, asociado a un estilo de programación OO el DBMSOO reduce la diferencia semántica entre el programa y la Base de Datos que lo soporta.**

Los objetos hacen también que las aplicaciones se ejecuten más rápido. Como los datos de una entidad están relacionados lógicamente, la BDOO tiene los medios para optimizar su localización física, las aplicaciones son capaces de leer menos archivos para recuperar todos los datos relevantes.

Los sistemas tradicionales de CAD, por ejemplo, almacenan a menudo cada componente de un diseño en un archivo separado y utilizan una base de datos para almacenar los nombres de los archivos. Para acceder al diseño compílo, la aplicación CAD debe abrir y cerrar todos los archivos de la lista de archivos. En una BDOO, el diseño entero con todos sus componentes puede almacenarse en un objeto, reduciendo grandemente el número de operaciones de archivo necesarias para acceder al diseño.

A diferencia de las BDR que sólo permiten conjuntos de datos del mismo tipo (char, date, long, double, etc.), las BDOO permiten conjuntos arbitrarios de objetos, dichos conjuntos de objetos pueden manipularse por el usuario o bien por la aplicación, bloqueados para gestión de transacciones, o agrupados para optimizar el rendimiento y facilidad de acceso.

No existe una definición estandar de lo que es una base de datos orientada a objetos, pero se ha establecido un primer concenso en la definición y conceptos

centrales referentes a los DBMSOO, descrito a través de las doce reglas de oro[#]. Por otro lado, se han construído comités que intentan actualmente sentar las bases de los estandares para varios aspectos de la tecnología de los DBMSOO.

Las reglas de oro descritas en [#] permiten identificar a un DBMSOO. Las primeras cinco por su funcionalidad en cuanto a DBMS y las siete restantes por su aplicación al paradigma de la **OO**.

Cinco reglas para definir un DBMS.

1.- Persistencia

2.- Administracion de Disco, buffer, índice, "clustering" de objetos y optimización de consultas.

3.- Integridad de Datos, Atomicidad de las transacciones (Commit, Rollback, abort), seguridad y confidencialidad.

4.- Control de concurrencia, varios usuarios, asegurar concurrencia de acceso sobre accesos, seriabilidad = noción mínima de concurrencia.

5.- Lenguaje de Consulta ad-hoc. En los DBMSR el SQL, ofrece mecanismos ad-hoc de consulta, sin usar el lenguaje de programación, herramientas gráficas, en los DBMSOO con un lenguaje de programación se genera un Query.

[#] M. Atkinson et al. "The Object-Oriented database System Manifiesto", Proceedings of the 1st Intl. Conf. on Deductive and Object-Oriented Databses, Kyoto Japan, December, 1989.

Siete reglas para definir un sistema orientado a objetos.

1.- Objetos complejos.

2.- Identidad de los objetos. Cada objeto posee un identificador que es independiente de su valor objeto = (identificador,Valor)

3.- Encapsulamiento de un Objeto = Interfaz (visible) + Implementación (invisible). Interfaz: operaciones permitidas sobre los objetos (métodos).

Implementación se da en dos partes:

- Datos (memoria del objeto)
- Operaciones (código de los métodos)

4.- Organización en tipos o Clases. Un tipo/clase describe las propiedades comunes de un conjunto de objetos:

- Estructura (datos).
- Comportamiento (especificación e implementacion de métodos).

5.- Herencia y gestalt.

6.- Sobrecarga y Ligado dinámico (overloading and late binding).

7.- Completez computacional[6]. No debe existir distinción entre el DML (ejemplo SQL) y el Lenguaje de programación.

Reglas de oro de un DBMSOO = OO + DBMS

[6] Esta característica exige que todas las opciones aplicables sobre los objetos no se realicen sino a través del lenguaje explícitamente especificado para ello y no haciendo uso de otras herramientas de menor nivel, para alterar la integridad de la base de datos.

Características opcionales

- Herencia múltiple. Generalización vs Especialización.
- Administración de versiones (la historia de los objetos).
- Transacciones largas y anidadas (Aplicaciones de CAD, trabajo cooperativo).
- Polimorfismo, genericidad (Type STACK of type T & T = Integer, real).
- Restricciones de integridad (¿Qué restricciones?, ¿cómo expresarlas?, almacenarlas, administrarlas y validarlas).

Opciones abiertas

Tipos y Clases, uniformidad del modelo.

El tipaje permite la limpieza del código: cada objeto, cada variable tiene un tipo.

Validación de tipos en tiempo de compilación o al momento de ejecución.

Generalmente los tipos no son Objetos.

Generalmente una clase tiene un método new y una extensión.

¿ Los métodos y las clases son objetos ?

Smalltalk, LISP O = si.

Eiffel, C++ = no.

Orion, Gemstone,Iris = si.

O2,Ontos = no.

OTRAS DEFINICIONES DE BASES DE DATOS ORIENTADAS A OBJETOS

James Martín define a una **base de datos orientada a objetos** como una **base de datos inteligente** que soporta el paradigma de la orientación a objetos, almacenando no sólo datos sino tambien sus métodos (procedimientos). Una BDOO está diseñada para ser eficaz, desde el punto de vista físico, para almacenar objetos complejos. Evita el acceso a los datos si ésto no se realiza mediante los métodos almacenados en ella, como lo muestra la siguiente figura:

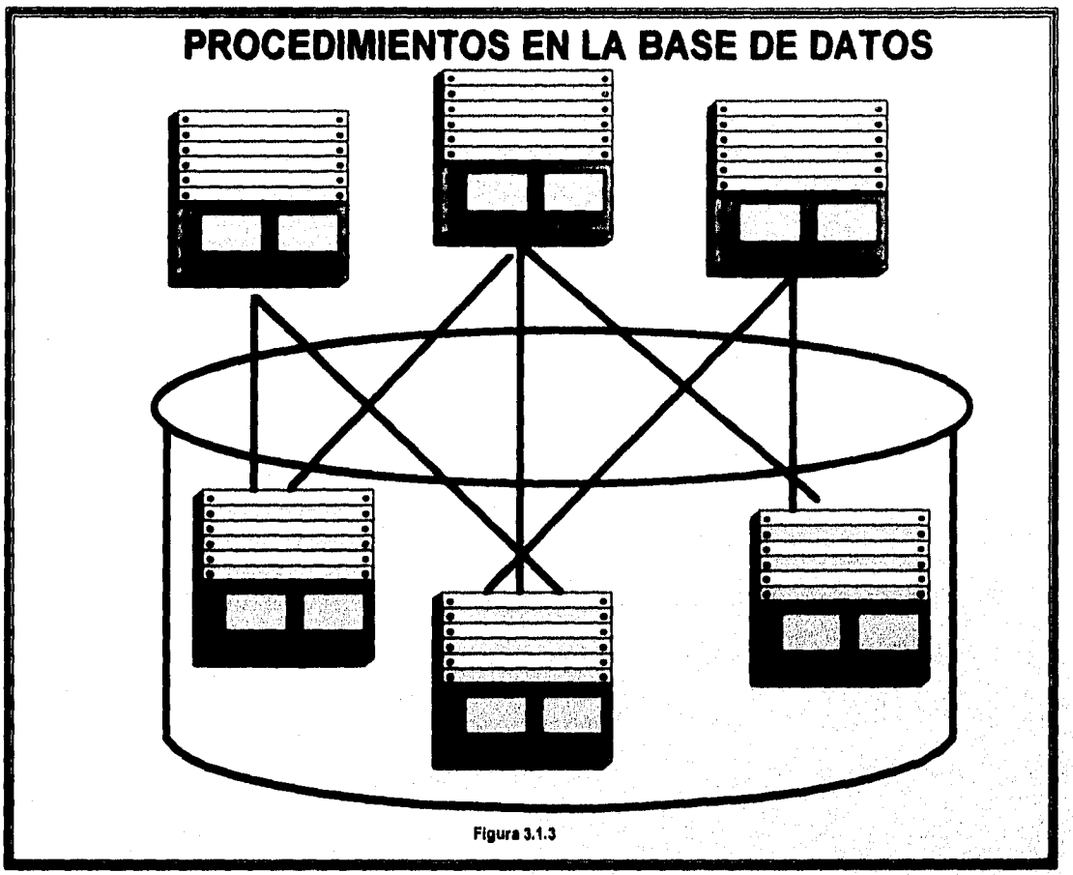


Figura 3.1.3

Las bases de datos orientada a objetos surgieron en un principio para soportar la programación orientada objetos. Los programadores de Smalltalk y C++ necesitaban almacenar lo que llamaban datos persistentes, datos que permanecen después de terminado un proceso. Las bases de datos se volvieron importantes para ciertos tipo de aplicaciones con datos complejos como CIM, CAD, CASE, CAP, también se volvieron importantes para el manejo de los BLOB (Binary Large Objects). Las aplicaciones como la organización de un periódico y la recuperación en video de las agencias de publicidad ya utilizan BDOO con BLOB. Las BDOO soportan tipos de datos más variados que las simples tablas, columnas y renglones de las bases de datos del modelo relacional. Las técnicas orientadas a objetos son buenas tanto para los sistemas CIM, CAD, CASE, CAP, como para el cómputo en general. Varias de las deficiencias en las bases de datos relacionales convergieron para formar una

nueva generación de sistemas para la administración y estructuración de los datos. Las bases de datos activas se implantan mejor con técnicas orientadas a objetos. Las bases con conocimiento que emplean los sistemas expertos se orientaron a los marcos, donde el marco es un objeto con un conjunto de reglas asociadas con él. Los tipos de datos abstractos (tipos de datos definidos por el usuario), necesitaban un soporte que incluyera: Groupware, audio, imagen y video para multimedia, y Hipermedia, Heurísticas, eventos para control de procesos, y administración de redes; estos datos complejos necesitaban técnicas de acceso que superaran el desempeño de las bases de datos por relación, como se muestra en la siguiente figura donde se observa la gama de necesidades que convergieron en la tecnología de las Bases de Datos Orientadas a Objetos.

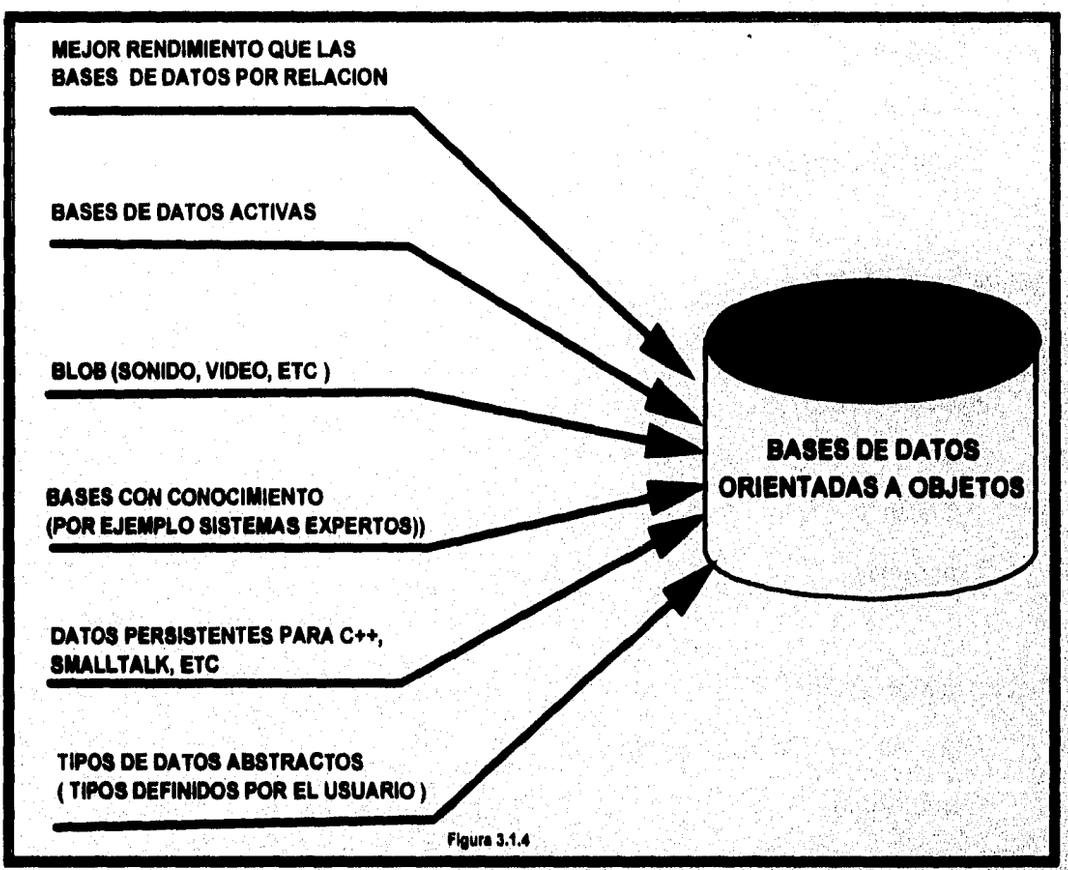


Figura 3.1.4

Nuevas aplicaciones

- Automatización de oficinas
- CAD/CAM
- Geografía y Cartografía
- Medicina
- Documentación
- Ingeniería de Software
- Reconocimiento y síntesis de voz
- Protección militar
- Publicidad
- Enciclopedias y diccionarios
- Educación
- Turismo
- Problemas Legales

Nuevos tipos de Datos

- **Textos**
- **Imágenes**
- **Voz**
- **Gráficas**
- **Programas**

Nuevas características

- **Gran volúmen de información**
- **Complejidad**
- **No estructurados o altamente estructurados**
- **Multimedia**

Diferentes clases de usuarios

Del usuario ingenuo al usuario especialista

Diferentes percepciones

- Modelado
- Interfaces
- Explotación, uso
- Mantenimiento

Tendencias actuales

- De los Datos: Hacia Objetos conocimientos, Objetos complejos, Multimedia
- De los DBMS Jerárquicos, de redes y relacionales hacia:

  DBMS Multimedia, generalizados

  DBMS Extensibles (Relacionales)

  DBMS Orientados a Objetos

3.2 FUNCIONALIDAD DE LAS BASES DE DATOS OO

Las BDOO ofrecen parte de la misma funcionalidad que los lenguajes de programación OO, ya que permiten la encapsulación de los métodos y datos dentro de la base de datos; activan métodos mediante mensajes a los objetos. Permiten la declaración de relaciones jerárquicas entre los objetos a través del uso de la herencia, además las BDOO ofrecen a los LPOO la persistencia (objetos que sobreviven al proceso que los creó) y compartición (control de concurrencia) de objetos de lo cual carecen la mayoría de los LPOO.

Con el transcurso de los años se ha presentado un movimiento consistente dentro de la técnología de las bases de datos para incorporar más semántica al modelo de datos. Si los datos reflejan de una manera más exacta el diseño de una aplicación, será más facil su disponibilidad para un usuario final, y se disminuye la necesidad de

comprender la forma en que se almacenan y manipulan. Las **BDOO extienden aún más la semántica de los datos.** Permitiendo en forma arbitraria la definición de tipos de datos complejos y proporcionando un medio para asociar el comportamiento con los datos, la semántica de las BDOO están más cerca de sus equivalentes del lenguaje de programación. Los LPOO y las BDOO son complementos naturales uno de otro a los lenguajes enfatizan el procesamiento, la estructuración compleja y los datos locales. Las BDOO se centran en un método más declarativo, datos compartidos fuera del dominio de las aplicaciones y soporte para grandes cantidades de datos, uno de los objetivos de los LPOO y de las BDOO es generar una función limpia entre ellos, conservando no obstante sus ventajas individuales.

Las BDOO difieren significativamente en funcionalidad con respecto a las BD Relacionales. Las BD-R se basan en derivar una estructura virtual en la ejecución basada en valores de los conjuntos de datos almacenados en tablas. Mientras que **las BDOO contienen objetos predefinidos que no necesitan derivarse en el momento de la ejecución.** En una BD-R las vistas se construyen seleccionando datos de múltiples tablas y cargándolos en una única tabla. **En una base de datos OO las vistas se obtienen pasando punteros de objeto en objeto.**

Una BDOO es una base de datos Activa mientras que una BD-R es pasiva, ya que las Relacionales ofrecen la posibilidad de agregar o borrar registros, **las BDOO ofrecen la capacidad de incorporar métodos dentro de los objetos, permitiendo así a la base de datos incorporar muchas de las operaciones que con una BD-Relacional deben dejarse a la aplicación.**

3.3 LOS ORIGENES DE LOS DBMSOO

Los siguientes tres puntos son las principales fuerzas que han motivado la evolución de las bases de datos orientadas a objetos (DBMSOOs):

1. **Mejor capacidad.** La motivación original fue la necesidad de manejar una mejor clase de DBMSs debido a las limitaciones de estructura y funcionamiento de los DBMS.

2. **Persistencia**. Con el incremento y disponibilidad de los lenguajes de programación orientados a objetos (LPOO), una segunda motivación fue la necesidad de almacenar y recuperar objetos después de la ejecución de un simple programa.

3. **Componentes reusables**.- finalmente como la programación orientada a objetos va madurando, esto incrementa la necesidad para tener depósitos comunes para contener clases reusables y otros componentes de software.

Los dos últimos usos de los DBMSOOs son algunas veces confusos, pero éstos son completamente distintos. Los usamos cuando deseamos un almacenamiento de objetos en forma persistente, los DBMSOO's almacenan actualmente instancias (datos) creados al correr alguna aplicación **OO**. Cuando es usado como componente repositorio el DBMSOO's almacena definiciones de clase que son usadas para construir nuevas aplicaciones.

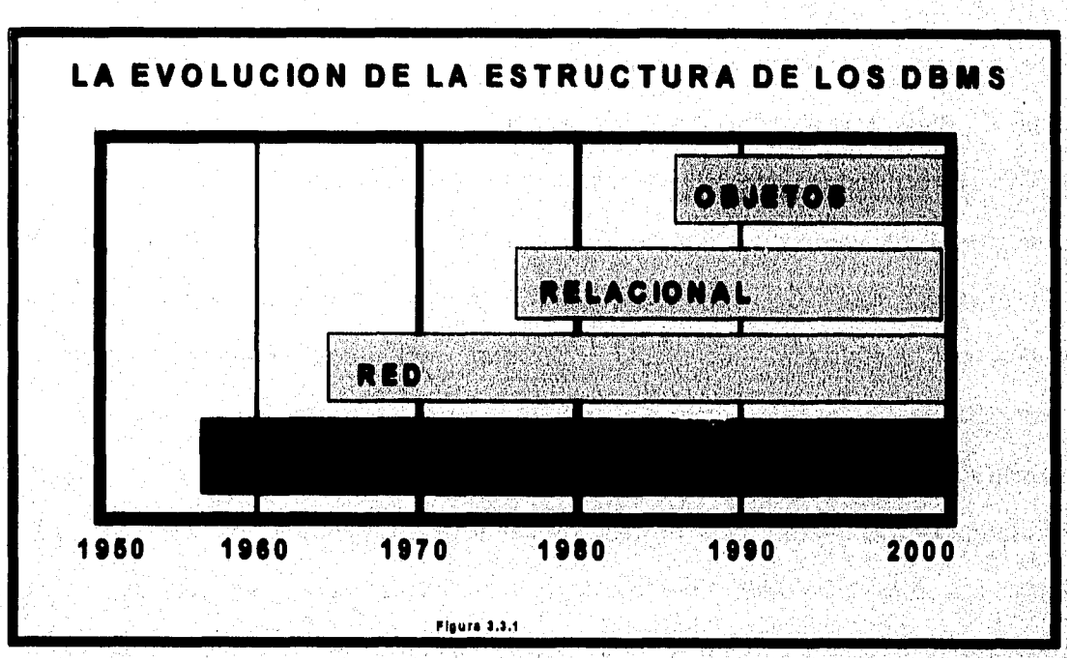


Figura 3.3.1

Estas tres necesidades son bien conocidas para crear la nueva generación de los DBMSOOs, como sea, existen diferentes dificultades con respecto a los tres puntos

anteriores. En las siguientes secciones veremos cómo cada una de estas diferentes necesidades es conocida por la tecnología de orientación a objetos.

Los DBMSOOs que actualmente se venden en el mercado fueron influídos por las tres motivaciones anteriores en varios grados. Para ayudar a escoger alguno de estos productos, menciono un número de ellos por nombre en el cuarto capítulo, una vez comprendidos los conceptos básicos que se darán en este capítulo.

3.4 CONCIBIENDO UN MEJOR DBMS

Si consideramos a un DBMSOO solamente como una nueva clase de DBMSs, se puede representar una repetición genuina en el almacenamiento de información. De hecho es un caso palpable que los DBMSOOs han combinando lo mejor de todas las generaciones precedentes de tecnología de administrador de datos.

En la siguiente figura se observa cómo los DBMSOO combinan lo mejor de sus predecesores.

Los DBMSOO's tienen le flexibilidad que se obtiene al manejar archivos planos. Igual que los archivos planos, los DBMSOOs le dan la libertad de almacenar cualquier clase de datos que se requieran, pero con la estructura y control de acceso que un DBMSs ofrece. Los DBMSOOs no restringen los tipos o tamaños de los datos que pueden ser almacenados y los usuarios son libres de inventar un tipo de dato de cualquier estructura. De hecho los DBMSOOs pueden almacenar cualquier cosa que pueda ser digitalizada, haciendola ideal para multimedia y otras aplicaciones avanzadas.

La estructura y velocidad de los DBMS-NETWORK.

Como los DBMS jerárquicos y de red, los DBMSOO proveen de ricas estructuras de datos con accesos rápidos pero sin la rigidez de estos. Las ricas estructuras son proveídas por objetos compuestos, árboles de herencia y otros objetos relacionales. **De hecho el modelo de datos orientado a objetos es un superset del jerárquico, de red y del modelo relecional**, así, este modelo OO garantiza el soporte de estructuras de complexión arbitraria. **El acceso rápido viene de que los objetos son unidos por**

un **identificador único IDs. continuo** mejor que llaves foráneas u otros mecanismos de relación. Los Ids de los objetos proveen un acceso directo a otros objetos, sin ellos sería necesario usar la tediosa búsqueda y comparación de operaciones.

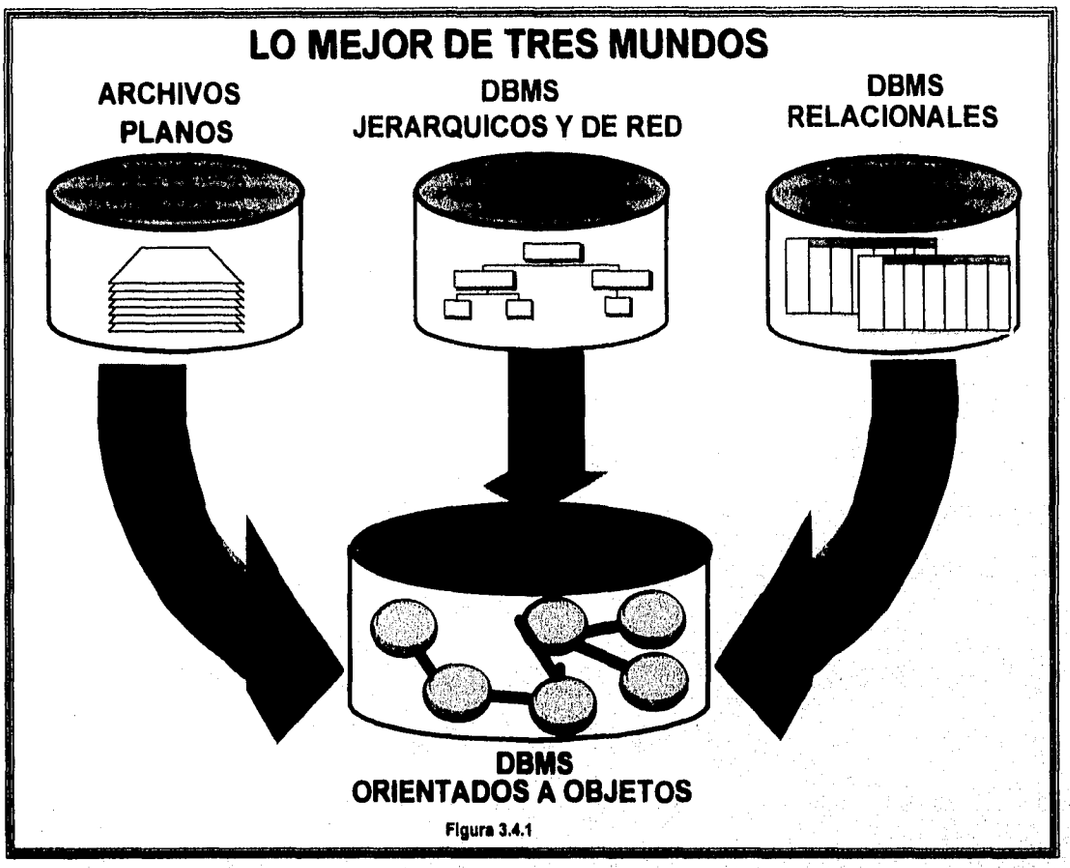


Figura 3.4.1

Al mismo tiempo los DBMSOOs proveen todas las flexibilidades de los DBMSs relacionales junto con poderosas capacidades para múltiples vistas de la misma información. Los DBMSOO con la flexibilidad de encapsulación en lugar de Hard-Wring con la información estructural dentro de los sistemas de bases de datos, como en los sistemas de red y jerárquicos. Esta información desempaquetada dentro de cada objeto individual, dándole esta encapsulamiento local y las estructuras de información pueden ser modificadas en el ambiente sin molestar otras aplicaciones que se estén ejecutando. Las múltiples vistas son posibles por objetos compuestos, las cuales combinan bajos niveles de objetos significativos y estructuras integradas

porque los objetos individuales pueden participar en cualquier número de objetos compuestos. No existe límite del número de "Views" que pueda ser creado por cualquier colección de datos. Además cada objeto compuesto contiene su actual Ids, estas vistas compuestas pueden ser usadas para realizar operaciones en la base de datos, incluyendo actualización de información.

Las bases de datos orientadas a objetos toman ventaja en las aplicaciones, en las que las relaciones entre los elementos de la base de datos llevan la información clave. Por otro lado, las bases de datos relacionales toman ventaja cuando los valores de los elementos de la base de datos llevan la información clave. Es decir el modelo orientado a objetos captura la estructura de los datos, mientras que el modelo relacional organiza los datos en sí. Si un registro pude ser comprendido aisladamente, entonces la base de datos relacional es adecuada probablemente. Pero **si un registro tiene sentido sólo en el contexto de otros registros, entonces una base de datos orientada a objetos es la más apropiada.**

3.5 PERSISTENCIA PARA LOS LENGUAJES DE PROGRAMACION ORIENTADOS A OBJETOS

En las sencillas aplicaciones de usuarios escritas con Smalltalk, el proveer de la persistencia de objetos no es un gran problema, ya que Smalttalk concluye cada sección salvando por completo la imagen de su ocurrente estado, incluyendo los valores de todas sus variables y sus objetos. Cuando Smalltalk es ejecutado nuevamente la persistencia de los objetos y variables es totalmente transparente.

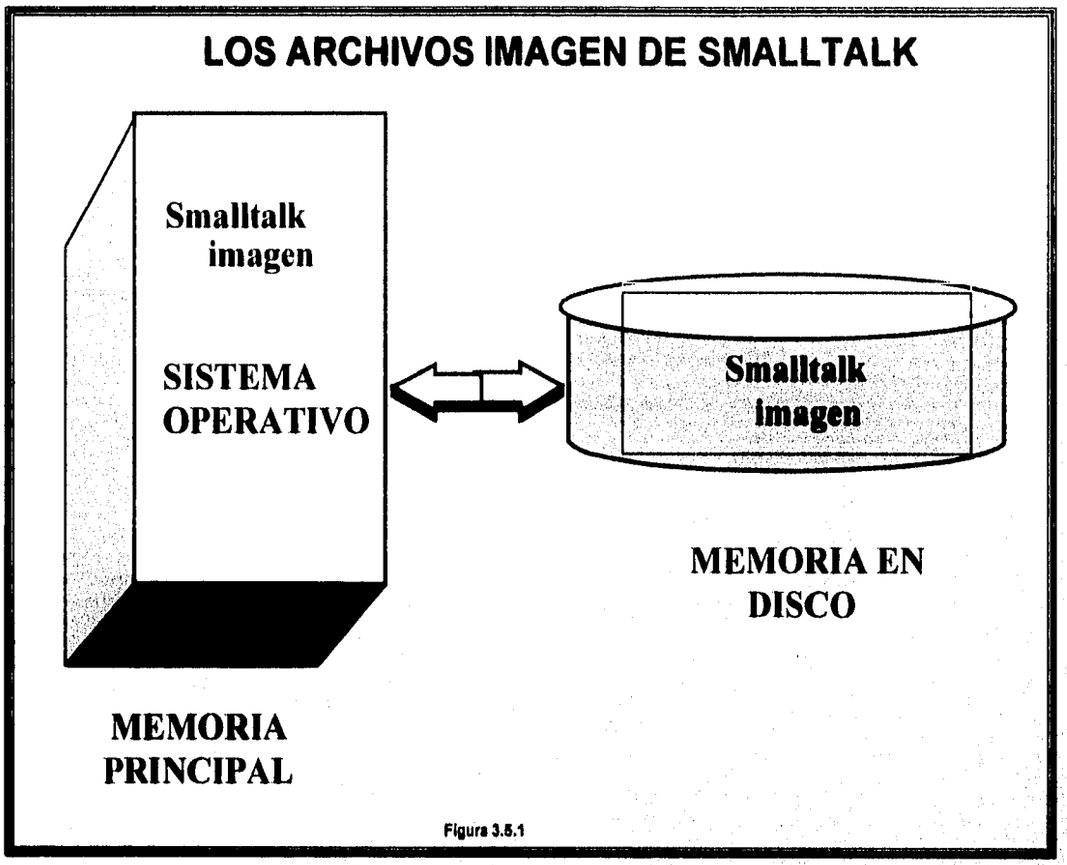


Figura 3.5.1

Para aplicaciones multiusuario es requerido algo más que la representación de archivos. Varias alternativas de representación de archivos han sido explorados. Es posible almacenar el objeto en archivos planos, pero los archivos planos son inadecuados para el acceso de los multiusuarios. Otra opción es almacenar objetos relacionados con DBMS Relacionales pero el funcionamiento con esta opción es generalmente de mala calidad, los objetos deben ser descompuestos en estructuras de registros sencillos, para adoptar el formato un sistema relacional al traer un objeto debe reconstruir la estructura.

La más obvia y viable alternativa es usar un DBMSOOs para almacenamiento y comparticion de objetos, construido exclusivamente para este propósito. Esto nos lleva a la segunda motivación para adoptar el DBMSOO.

Este surgimiento del uso de DBMSOOs ha colocado nuevas demandas en sus diseños considerando que la razón inicial fue dirigida por las empresas de DBMSs;

esta segunda tiene interés de manejar lenguajes propietarios de DDL y DML. Con este cambio el acceso tradicional al usar un propietario de DDL/DML llegó a ser menos atractivo porque éste requería una traslación entre el lenguaje de aplicación y el lenguaje de la base de datos con cada transacción.

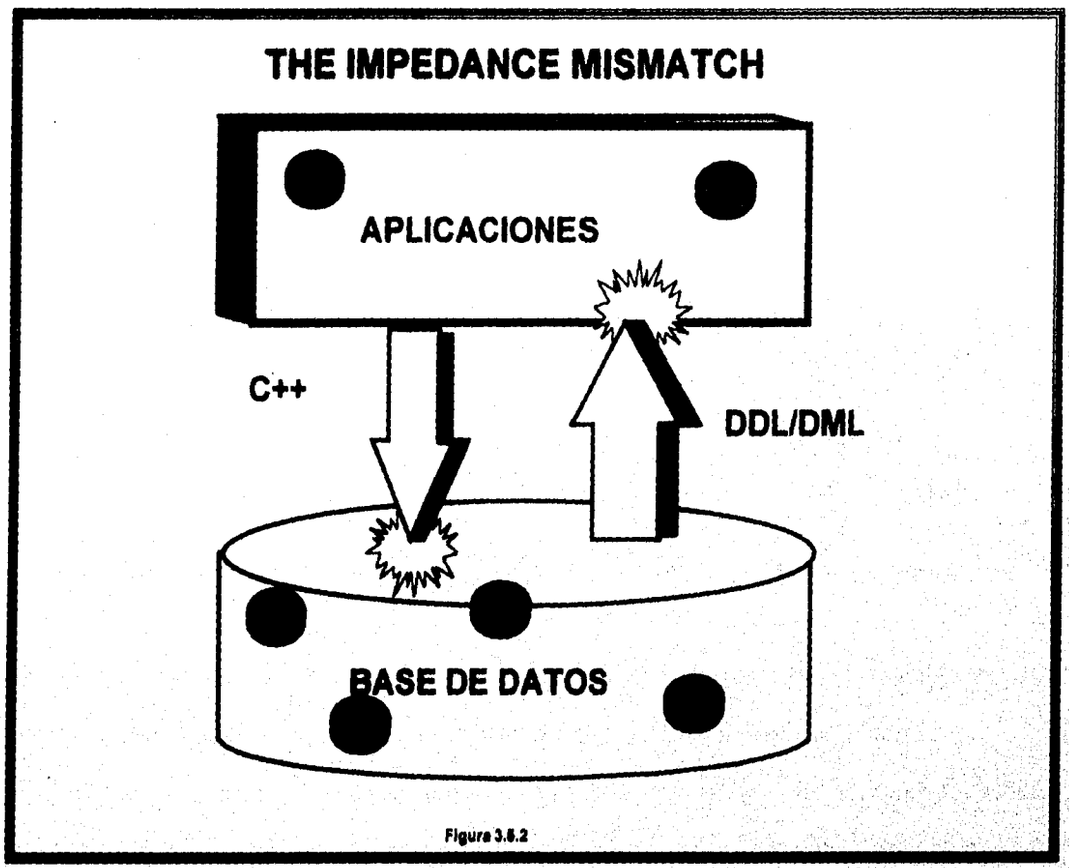


Figura 3.6.2

Por ejemplo, supongamos que las aplicaciones son programadas en C++. Trasladando los objetos de C++ a Opal o COP, los objetos requieren un paso extra por parte del programador, uno que disminuya ambos desarrollos y mejore el funcionamiento al momento de ejecutarse. Este mal conlleva a una segunda opción de DBMSOOs que fueron diseñados como extensiones de lenguajes existentes.

La alternativa más común fue C++ con LISP entrando como un segundo distante.

De acuerdo con esta tendencia, Ontologic retira su producto Vbase del mercado y lo remplaza con un C++ compatible con DBMSs llamado Ontos.Servio con Smalltalk

igual que Opal, continuando con el soporte de interfase Smalltalk pero también añadiendo una interface de C++.

Dándole las nuevas variaciones acerca de la compatibilidad del lenguaje, algunos de los nuevos principiantes en el campo de los DBMSOOs estuvieron un poco menos preocupados acerca de la funcionalidad de los productos de DBMS, por ejemplo Object Design's Object Store proveé muy rápido y eficiente el acceso a los objetos persistentes C++ a través de una combinación de memoria virtual. Mientras que este planteamiento ha realizado tremendas ventajas, la idea de obtener más beneficios al programar con C++ va ganando adeptos en los vendedores de este tipo de bases de datos. Mediante ésto se apuntaría directo al admiración de terror en los corazones de la mayoría de los administradores tradicionales de base de datos.

3.6 CONTENEDORES DE COMPONENTES DE SOFTWARE

Como el mercado de la tecnología de orientación a objetos continúa madurando, motiva aún más el surgimiento de los DBMSOOs, ya que el software del futuro será totalmente construido a partir de librerías de clases reusables. debe haber algunas depósitos (almacenes) para almacenar aquellas clases y repetirlas entre muchos programadores.

La solución más rápida a este problema fue usar el mismo código fuente del sistema de control con el que se trabajaba para almacenar funciones para programación procedual.

Este planteamiento fue adecuado para pequeños sistemas, pero la gran escala de reusabilidad del software requiere un sofisticado control de concurrencia, exhaustos checkouts, versiones del sistema y otras características de los tradicionales DBMSs.

Los DBMSOOs representa la obvia alternativa para almacenamiento y distribución de clases reusables.

Estas nuevos productos de DBMSOOs amenaza con tener otro requerimiento de tecnología: la habilidad para almacenar objetos escritos en diferentes lenguajes.

Hasta el día en que se escribe ésta tesis, los contenedores de objetos requieren que todos los objetos sean generados en el mismo lenguaje de programación. Mientras que toman poder, puede ser relativamente locales los esfuerzos del desarrollo, esto parece improbable en los esfuerzos por estandarizar un LPOO.

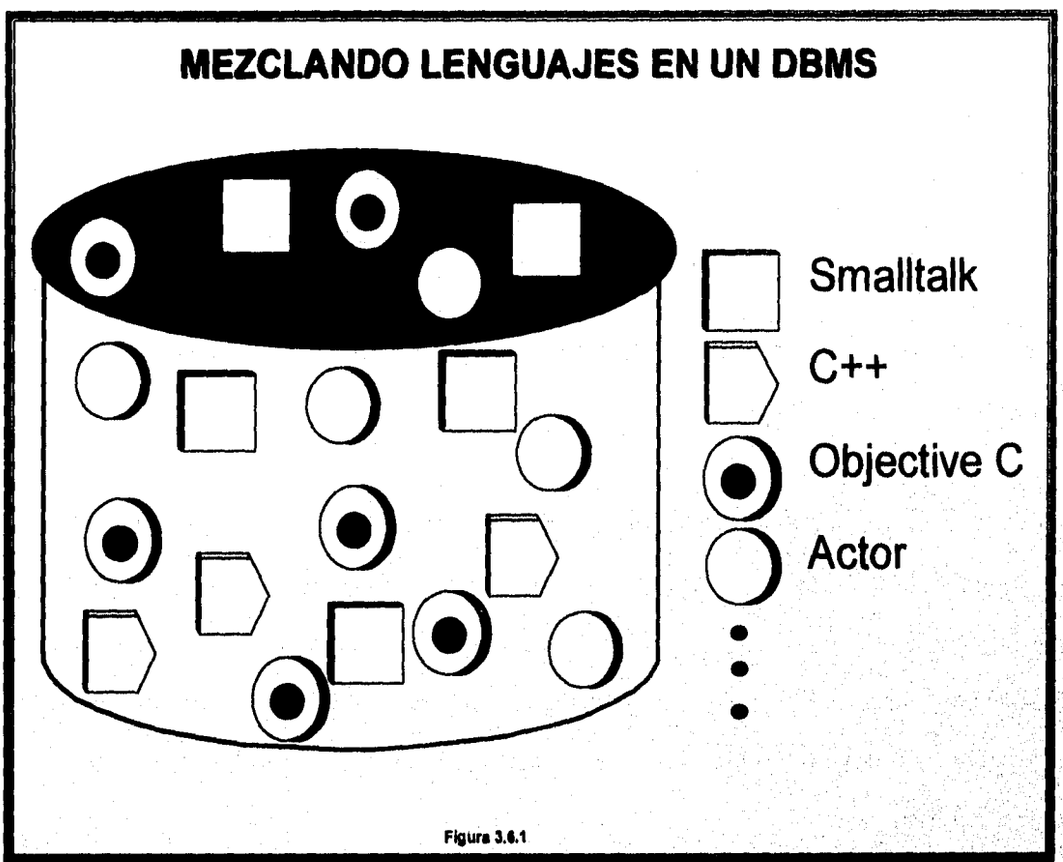


Figura 3.6.1

En el presente, algunos productos soportan esta opción, como Servicion Gemstones por ejemplo, que almacena todas las clases con su propio formato, **Opal** ejecuta algunas transacsiones con "**Smanltalk**" o "**C++**" para almacenar y recuperar objetos; en contraste, el sistema "**Versant**" almacena los objetos con C++ y Smalntalk en forma similar a su formato nativo. Como quiera que sea, cada tipo de objetos cualquiera puede ser acsesado solamente en su forma original, o en su defecto, desde el lenguaje C.

Finalmente, largos contenedores de componentes reusables tendrán que proveer algo más que los browser prototipos típicos de Smalltalk, para ayudar a los programadores a encontrar sus clases. Aunque los browsers son auxiliares y simplemente muy lentos e inmanejables cuando cientos de clases pueden ser escaneados y revisados. Los programadores necesitan asistencias activas para sus contenedores y encontrar la clase que ellos necesitan.

El contenedor ideal para un programador puede enlistar un set de clases disponibles que a el le interesen. Asi el programador puede decidir cuando quiere usar la clase existente, la subclase de una de las existentes, o crear una nueva linea de partida.

3.7 ALMACENAMIENTO DE OBJETOS COMO ELEMENTOS DE DATOS

Una de las limitaciones del modelo relacional, citada con mayor frecuencia, es su descripción de las entidades del mundo real a través de estructuras "planas", con muy pobre riqueza semántica (fidelidad en la descripcion de los objetos de la realidad). Es decir, los modelos orientados a objetos se construyen a partir de estructuras de datos que no estan en Primera Forma Normal, gracias al manejo de variables de instancia (atributos) no atómicos, sino de tipo tupla, conjunto, lista o resultado de una aplicación ortogonal de tales constructores.

El almacenamiento de objetos como elementos de datos cambia la realización de otras características que todas las bases de datos deben proporcionar. Como los objetos pueden transferirse eficazmente en una sola operación, es más fácil desarrollar algoritmos de Caché de disco para transferir grupos lógicos de objetos,

OBJETOS COMPLEJOS

Un objeto complejo es un objeto que a su vez está compuesto de otros objetos, los cuales a su vez estan compuestos de más objetos, etcétera. Debido a esto, es frecuente que la estructura de la clase sea muy compleja. Al utilizar un objeto sus datos, se deben leer sin tener que mover el mecanismo de acceso al disco. Los

datos de cada objeto deben estar unidos, lo cual no ocurre si se utiliza una base de datos por relación. El objeto podría necesitar varios datos de varias relaciones (tablas), por lo que habría que mover el mecanismo de acceso a disco para leer la otra tabla o relación. Los DBMSOO evitan deliberadamente el modelo de relacional con el fin de mejorar el rendimiento de la máquina, permiten que los datos de una clase no lleguen entre sí de manera eficiente; con frecuencia recurren a la estrutura de apuntadores que las BD Relacionales procuran evitar. Si una clase utiliza con frecuencia determinados métodos, tiene sentido optimizar la estructura de datos para mejorar el rendimiento de la máquina mediante tales métodos.

Un objeto complejo es una instancia de la jerarquía de la composición de las clases. Cada objeto tiene un conjunto de atributos de instancias y métodos. El valor de un atributo puede ser un objeto o un conjunto de objetos. **Los objetos complejos pueden definirse como agregación de otros objetos.** Un objeto implica atributos y métodos. Al utilizar un objeto sus datos se deberán leer sin tener que mover el mecanismo de acceso al disco.

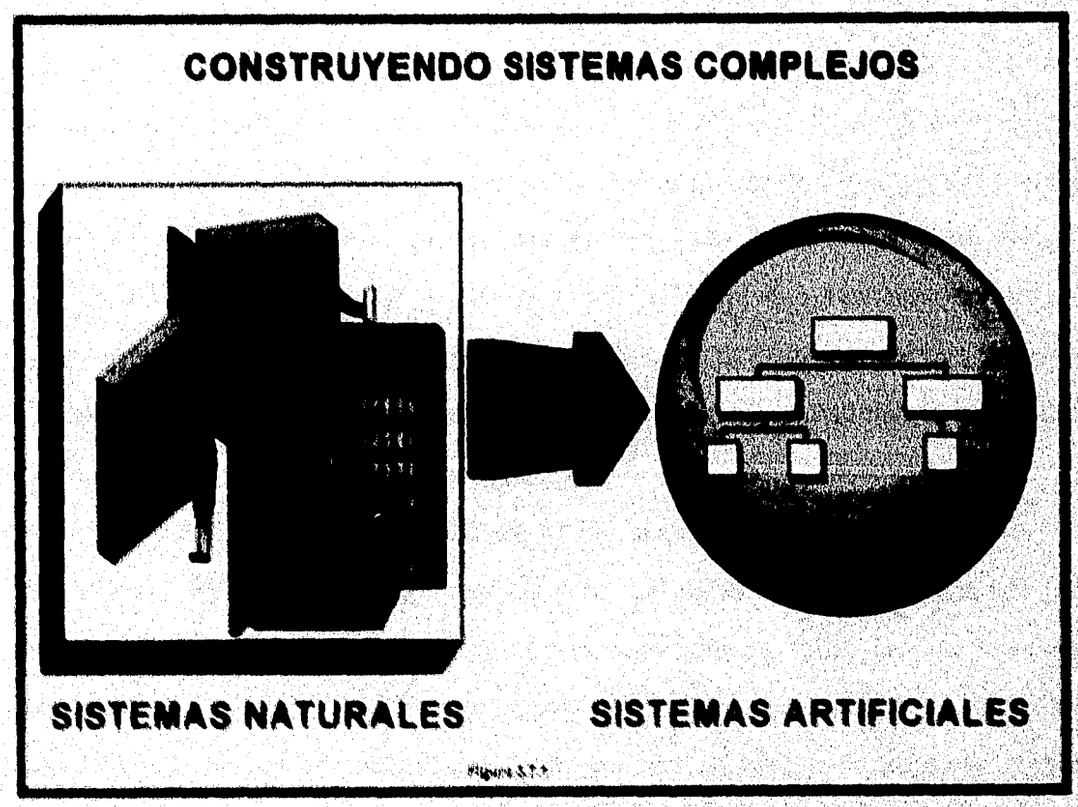


Figura 3.7.3

datos de cada objeto deben estar unidos, lo cual no ocurre si se utiliza una base de datos por relación. El objeto podría necesitar varios datos de varias relaciones (tablas), por lo que habría que mover el mecanismo de acceso a disco para leer la otra tabla o relación. Los DBMSOO evitan deliberadamente el modelo de relacional con el fin de mejorar el rendimiento de la máquina, permiten que los datos de una clase se liguen entre sí de manera eficiente; con frecuencia recurren a la estrutura de apuntadores que las BD Relacionales procuran evitar. Si una clase utiliza con frecuencia determinados métodos, tiene sentido optimizar la estructura de datos para mejorar el rendimiento de la máquina mediante tales métodos.

Un objeto complejo es una instancia de la jerarquía de la composición de las clases. Cada objeto tiene un conjunto de atributos de instancias y métodos. El valor de un atributo puede ser un objeto o un conjunto de objetos. **Los objetos complejos pueden definirse como agregación de otros objetos.** Un objeto implica atributos y métodos. Al utilizar un objeto sus datos se deberán leer sin tener que mover el mecanismo de acceso al disco.

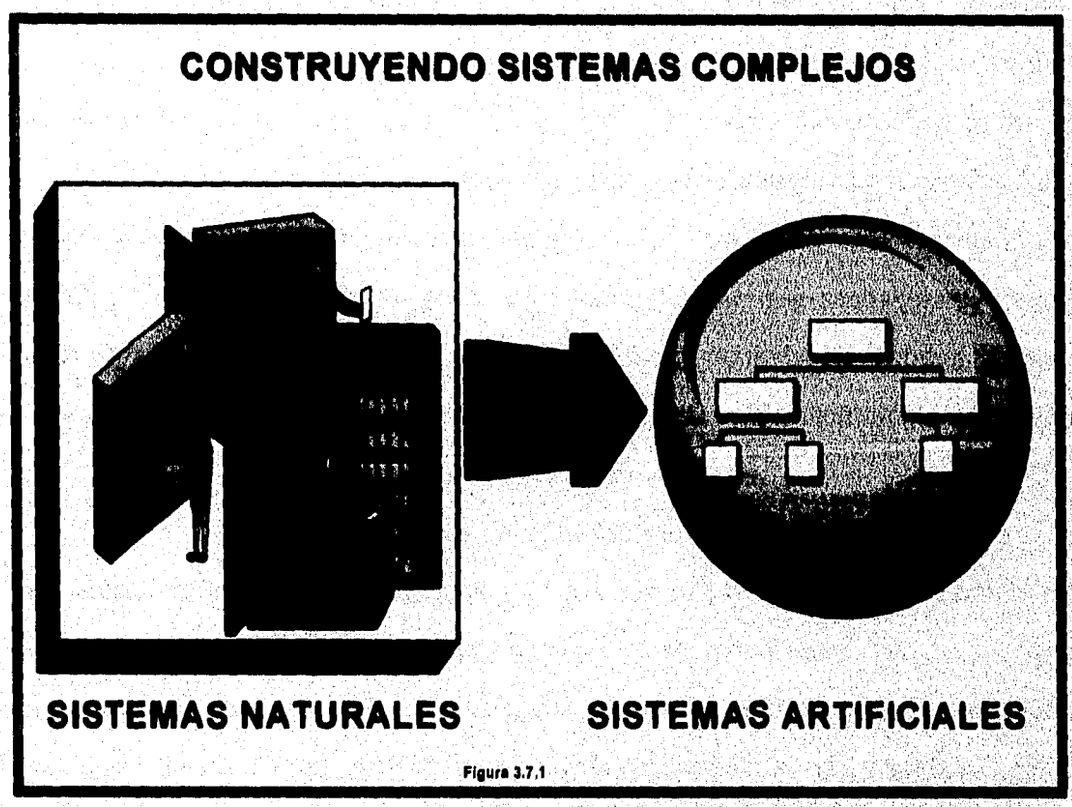


Figura 3.7.1

ESTRUCTURAS PARA ALAMACENAR OBJETOS

Diferentes constructores pueden utilizarse en la definición de objetos complejos y sus valores. Los OODBMS proporcionan un conjunto mínimo de constructores que incluyen :

Conjuntos	**(Sets)**
Listas	**(lists)**
tuplas	**(tuples)**

Un objeto complejo puede referenciar cualquier número de otros objetos de manera recursiva.

Una relación importante utilizada en el concepto de objeto complejo es la relación, "es parte de" (agregación), es decir que un objeto es parte de otro objeto.

Si la raíz de un objeto es borrada, todos sus objetos componentes son borrados.

En algunos Sistemas un candado en la raíz de un objeto complejo es propagado hacia todos sus componentes.

PERSISTENCIA

Se dice que un objeto es persistente si continúa existiendo aún después de que ha terminado el programa de aplicación. Normalmente los objetos que se existen durante la ejecución de un programa orientado a objetos surgen por medio de una de estas tres formas:

1. El objeto es creado estática o dinámicamente por el programa.
2. El objeto se recupera a partir de un archivo plano o una serie de tablas de una base de datos relacional.
3. El objeto se recupera de una base de datos orientada a objetos y está listo para su uso.

Los objetos persistentes contienen datos que permanecen durante más tiempo que el que emplea el programa de aplicación. A diferencia de aplicaciones construidas a partir de lenguajes orientados a objetos[Θ], en las que los objetos se crean durante la ejecución y finalizan con la sesión de la aplicación, los objetos de una BDOO sobreviven múltiples seciones.

También contrastando con las tablas o bases de datos relacionales, los datos de los objetos persistentes no modifican su formato aunque pueden moverse entre las aplicaciones y los datos en que residen permanentemente.

La persistencia de un objeto está íntimamente ligada al concepto de identidad de un objeto. La persistencia es la primera característica que ofrece todo DBMS.

Persistencia es una funcionalidad requerida y es una parte importante obviamente de un DBMS, en los DBMSOO la persistencia debe ser:

1. **Ortogonal a los tipos:** Cualquier dato, cualquier tipo puede ser persistente en cualquier momento.
2. **Trasnsparente:** Los datos persistentes y no persistentes se deben manipular exactamente igual, desde el punto de vista del usuario.
3. **Independiente:** El DBMSOO debe proveer automáticamente operaciones explícitas para lectura y escritura de los objetos desde y hacia el disco para efectuar el mantenimiento de la integridad de la información.

El problema principal está en que la estructura de los objetos puede diferenciarse en mucho en cada instancia de una clase y ésto no es muy adecuado para organizarlas en la estructura física del disco (archivos, páginas, bloques físicos, registros, etc.).

Deben existir mecanismos para definir qué objetos son persistentes y cuales no.

IDENTIDAD DE LOS OBJETOS

Cada entidad del mundo real es modelado por un tipo de objeto, y a cada objeto se le asocia un único e invariable identificador, el cual será almacenado como una tupla de los identificadores de sus partes, e independiente de la forma en que el objeto

[Θ] A excepción de Smalltalk

pudiese cambiar la identidad del objeto que permita un proceso de relaciones complejas. Si los objetos se refieren entre sí por medio de sus únicos identificadores, sus relaciones continuarán aún cuando el objeto cambie su estado, valor o situación. La identidad del objeto contrasta con las BDR basadas en valores en las que las entidades se identifican por sus atributos y por ello pueden cambiar a lo largo del tiempo, dicho de otra forma, la identidad de un objeto es esencialmente útil en el mantenimiento de la persistencia de las relaciones de los objetos.

En los DBMSOO un objeto es identificado exclusivamente por un identificador único (IDO), el cual es independiente del estado y/o valores del objeto. Con los IDO los objetos pueden compartir subobjetos y brindan la capacidad de construir redes de objetos generales. Además con los IDO se introducen dos tipos de igualdad entre objetos.

- **Igualdad de identidad.**
- **Igualdad de valores.**

La idea de identidad en Bases de Datos Orientadas a Objetos es que cada objeto tiene una existencia que es totalmente independiente de sus valores o de su estado. Dos objetos pueden ser idénticos (ellos son el mismo objeto o tienen la misma identidad) o pueden ser iguales (ellos tienen los mismos valores). Los objetos idénticos pueden distinguirse de los objetos iguales en dos formas o aspectos: Compartición de objetos y actualización de objetos.

Existen tres alternativas para generar los IDO:

1. Independiente.- Se generan independientemente del ambiente (Gem-Stone) Lleva un control de los objetos creados y luego hace referencia a través de técnicas Hash.

2. Lógico.- Los Ids contienen el número de la clase y el número de la instancia (Orion). Esta alternativa tiene el problema de que la clase del objeto no puede cambiar.

3. Físico.- Los IDO se derivan de la dirección del objeto en disco (O^2) y el nombre de la clase se almacena con un atributo del objeto. No es muy óptimo ya que se tiene que leer el objeto completo para saber su clase y ejecutar su método.

COMPARTICION DE OBJETOS

En un modelo basado en identidad, dos objetos pueden compartir componentes, y su representación gráfica es un **grafo.** Tómese el siguiente ejemplo donde una persona tiene un nombre, edad y un conjunto de hijos. Araceli y Pedro tienen un hijo Kevin de 3 años. En la vida real, los dos casos se pueden dar, que Pedro y Araceli tengan un mismo hijo o bien cada uno de ellos tienen su propio hijo. En un sistema sin identidad Araceli y Pedro se pueden representar como:

(Araceli,29,{(Kevin,3,{})}

(Pedro,31,{(Kevin,3,{})}

En este caso no hay una forma de representar si Pedro y Araceli son los padres del mismo hijo. En un modelo basado en identidad, si es posible que dos estructuras compartan un mismo objeto, y además es posible representar las dos situaciones:

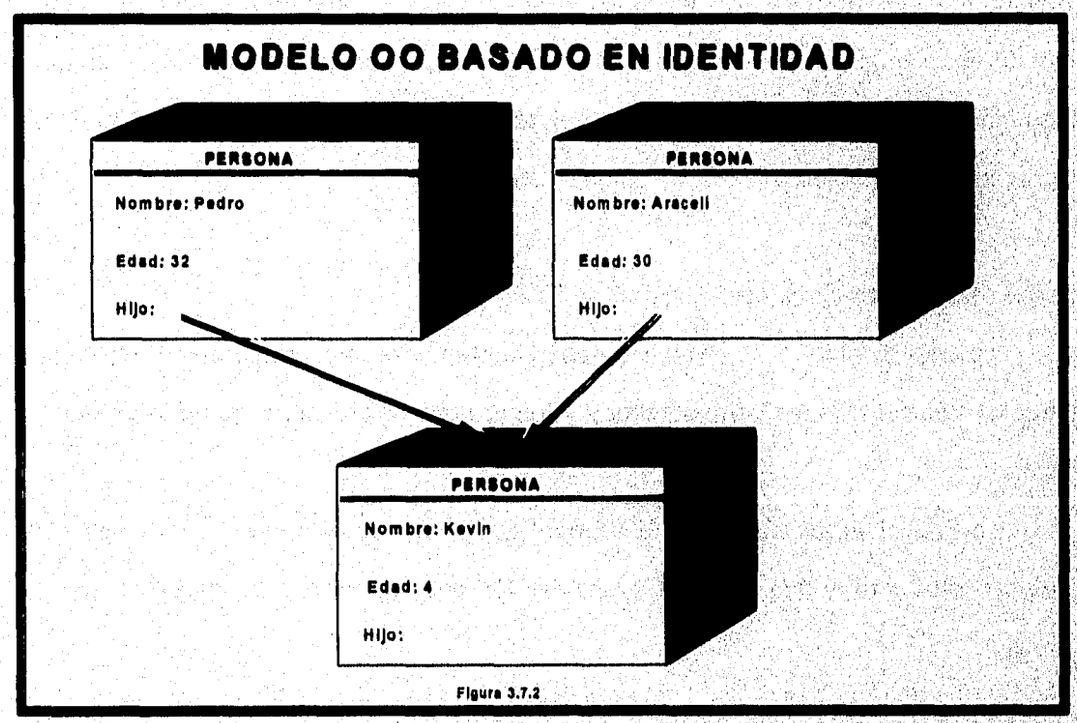


Figura 3.7.2

En los DBMSOO cada objeto tiene una identidad única, y no es posible que dos objetos diferentes tengan una misma identidad. El manejo de objetos complejos o el poder compartir objetos por dos o más estructuras no es una característica exclusiva de los DBMSOO, también es posible hacerlo por medio de un DBMSR o de llaves foráneas, sin embargo en los DBMSOO la identidad es un mecanismo transparente para el usuario, ya que el DBMSOO es el encargado de definirlo y utilizarlo, mientras que en un DBMSR el usuario es el responsable del manejo y definición de los identificadores de cada tupla a través de llaves primarias.

ACTUALIZACION DE OBJETOS

Si se asume un modelo de identidad por valor, cuando se desee hacer una actualización del hijo Kevin, en el caso de que el sea hijo de Araceli y Pedro, la actualización deberá hacerse en los dos casos, es decir los dos subobjetos deberán actualizarse por separado.

Esta situación no pasa en un modelo de identidad de objetos, bastaría que Kevin se actualizara por Pedro o por Araceli, ya que se comparte su referencia por los dos padres. El DBMSOO es el que se encarga de encontrar a Kevin y actualizarlo.

ALGUNAS FACILIDADES DE LA IDENTIDAD DE OBJETOS

La identidad de objetos es una herramienta primitiva muy poderosa que permite la construcción de conjuntos (SETS), tuplas, y manipulación de objetos complejos en forma recursiva.

El tener un manejo primitivo de identidad de Objetos por el manejador de la base de datos, implica ofrecer operaciones tales como:

Asignación de objetos.

Copiado de objetos (A profundidad a un sólo nivel).

Pruebas de identidad (Objetos idénticos) .

Pruebas de igualdad (Valores iguales a profundidad a un sólo nivel).

Por otra parte se puede simular identidad de objetos con sistemas basados en valores, a través del uso de identificadores explícitos de objetos, sin embargo el usuario es el responsable de su manejo. No obstante esta alternativa, no asegura que un identificador sea único y mantenga la integridad referencial.

Los modelos basados en identidad son la norma en los Lenguajes imperativos LPOO puros. Cada objeto manipulado en un programa tiene una identidad y pude ser actualizado. Esta identidad puede venir del nombre de una variable, o bien de una dirección física en memoria principal o secundaria, en el caso de ser persistente. Este esquema de identidad es muy nuevo y no es una característica que pueda ser encontrada en un DBMSR.

ENCAPSULAMIENTO Y MÉTODOS

Uno de los objetivos de la tecnología tradicional, de las bases de datos es la independencia de datos. Las estructuras de datos deben ser independientes de los procesos que utilizan los datos, asi los datos se pueden utilizar en la forma en que deseen los usuarios. En contraste con lo anterior, una de sus principales premisas de la técnologia OO es la encapsulacion; esto significa que los datos sólo pueden ser utilizados por los métodos (código) que forman parte de una clase. Se pretende que las clases de la OO sean reutilizables por lo que otra premisa de la OO es lograr la máxima reutilzación del código. Por lo que la clase debe estar libre de errores y sólo debe modificarse en caso absolutamente necesario. La tecnología de los DBMS tradicionales esta diseñada para soportar procesos sujetos a una modificación infinita, Por ello es necesaria la independencia de los datos. Los DBMSDOO soportan clases, las cuales rara vez son modificadas, los cambios provienen de la herencia múltiple de clases o de las clases ligadas entre sí de varias formas. Por lo que las estructuras de datos en las BDOO deben estar completamente optimizadas para soportar la clase en la que estan encapsuladas. Por ello, los objetivos en las BD relacionales y las BDOO son totalmente diferentes.

Cada objeto en una BDOO puede tener encapsulados en su interior un número de métodos para actuar sobre sus datos, estos métodos son activados cuando el programa de la aplicación envía mensajes al objeto. Continuando con el ejemplo visto al inicio del capítulo, un objeto estudiante podría contener un método para determinar el promedio de calificaciones del estudiante. El método calcula el promedio de calificaciones del estudiante enviando el mensaje **obtener calificación** a cada uno de los objetos del curso del estudiante, promedia las respuestas y devuelve el resultado. Cualquier programa de aplicación que requiera el promedio de algún estudiante sólo tiene que enviar el mensaje **promedio** al objeto estudiante, en lugar de realizar el cálculo por sí mismo. La encapsulación de métodos y datos elimina buena parte del trabajo de programación para el desarrollador, desviando la responsabilidad de recuperación y actualización de la base de datos hacia la propia base de datos.

Aunque el software de aplicación puede realizar la comprobación de la integridad de los datos, es más fácil y menos propenso a error dejar que la base de datos se encargue de ello. Por ejemplo determinados cursos pueden tener prerrequisitos, inscripción limitada y otras limitaciones, si el software de aplicación es responsable de comprobar estas limitaciones, entonces toda aplicación que añada estudiantes a los cursos debe contener código para comprobar esas limitaciones. Y cada vez que cambien las limitaciones, todos los programas de aplicación deben actualizarse. En comparación con esto, una base de datos orientada a objetos puede encapsular métodos que comprueben las limitaciones con los mismos objetos del curso, esto garantiza que las limitaciones del curso serán satisfechas cada vez que en un programa de aplicación acceda a los cursos. Además, un método orientado a objetos localiza el código de limitación de forma que es fácil actualizar la base de datos para reflejar los cambios en las limitaciones.

Sin embargo, la encapsulación es limitativa desde el punto de vista de base de datos, ya que exige la especificación de métodos especializados a cada tipo de

consulta, perdiendo con ello la flexibilidad tradicional ofrecida en los tradicionales DBMS.

HERENCIA

Con las BDR, los datos se normalizan para evitar la redundancia y las anomalías provocadas por la redundancia de los datos, lo cual evita la redundancia en los datos más no en el código de la aplicación.

La Tecnología OO utiliza la herencia para reducir el desarrollo redundante de los métodos, también crea clases diseñadas para utilizarse otra vez en muchas aplicaciones. El encapsulado y la herencia reducen la cantidad de código redundante; asi como **también reducen los datos redundantes** de dos maneras:

1. Herencia.
2. Reutilizacion de clases.

Lo que permite reducir los costos del desarrollo y mantenimiento.

Con la tecnología de DBMS tradicionales el procesaminento común se realiza a través de distintas aplicaciones, pocas veces se llega a aprovechar el código, lo que provoca una menor productividad y mayores costo de desarrollo y mantenimiento. De manera más reciente las BD Relacionales han introducido el concepto de **reglas de activación** o procedimientos que puede ejecutar la base de datos. El uso de estas capacidades para colocar el procesamiento común dentro de la base de datos es un avance, sin embargo sigue separado el procesamiento de aquellas entidades naturalmente asociadas con el procesamiento. Por ejemplo, uno podría escribir una regla de activación que impusiera el comportamiento donde el salario de un empleado despedido fuese $0. Sin embargo, el proceso de despido no se puede asociar a la tabla "empleado".

En un DBMSOO este proceso es directo, ya que la clase "empleado" definiría la solicitud **despedir** e impondría la restricción de salario = $0.

El mecanismo de la herencia es de gran utilidad en el modelado de una aplicación. De hecho está también presente en otros modelos (extenciones al modelo relacional y modelado semántico), a través de los mecanismos de abstracción, especialización y generalización, muy frecuentemente usados en el modelado de las entidades de una aplicación. La herencia proporciona una herramienta poderosa ya que reduce el esfuerzo necesario para desarrollar y brindar el mantenimiento necesario a un sistema de base de datos. Pero esta capacidad de permitir que una base de datos envejezca graciosamente es más importante. Cuando se utiliza adecuadamente, la herencia permite que se añadan nuevas características y tipos de datos a una base de datos exigiendo solamente unos cambios muy concretos.

Una base de datos relacional tiene únicamente un conjunto limitado de tipos de datos
incorporados, por ejemplo: CHARACTER, INTEGER, LONG, FLOAT, DOUBLE, SERIAL, FECHA, ETC.

Un conjunto limitado de operaciones incorporadas (por ejemplo sum, count, etc., un diseñador de bases de datos relacionales) puede crear estructuras de datos más complejas combinando linealmente tipos básicos, como los campos en registros:

Estudiante =

Nombre	Edad	Dirección

Desgraciadamente, como no hay forma de añadir nuevas operaciones para estos nuevos tipos, las operaciones sobre ellos están restringidas hacia las definidas para tipos básicos. Una base de datos orientada a objetos proporciona un conjunto equivalente de tipos y operaciones incorporados.

La herencia es una herramienta de programación tan eficiente que no sólo ayuda en el proceso a definir los tipos de datos y de relaciones entre los datos a almacenar en una base de datos, sino que también reduce el esfuerzo necesario para efectuar los cambios necesarios en la estructura de la base de datos. El agregar un nuevo tipo de datos se realiza fácilmente por medio de la subclasificación, con la garantía de que

los tipos de datos y métodos ya existentes no serán afectados por el cambio. Al modificar la representación interna de un tipo de dato ó método, podría hacerse en aquella clase de la jerarquía de clases en la que se define el tipo de datos o método. Finalmente, debido que los cambios en la estructura jerárquica de la base de datos continúan para asociar los datos que están relacionados lógicamente, el mecanismo de la herencia soporta funciones de base de datos como:

- Bloqueo
- Autorización
- Consulta

La jerarquia de clase determina una trayectoria de herencia entre la superclase y sus subclases, permitiendo al diseñador reutilizar las definiciones de clases ya existentes incorporando sus atributos y métodos.

La jerarquia de composición (generada al permitir asignar como valor de un atributo el identificador de otro objeto) es la base de la definición recursiva de un objeto complejo en términos de otros objetos.

BIBLIOTECA DE CLASES

Una clase representa un molde o patrón de un conjunto de objetos similares, el usuario define las clases a partir de las bibliotecas de clases predefinidas que son parte del DBMSOO, las bibliotecas de clases permiten a los programadores **reutilizar,** en lugar de tener que volver a crear las estructuras de datos más comúnmente necesarias, como por ejemplo: matrices, diccionarios, tablas y fechas; las bibliotecas de clase incluyen también métodos, como recorrido lineal o dispersión (Hasshing). Al igual que con un LPOO, los programadores de BDOO perfeccionan las bibliotecas de clases y las jerarquías añadiendo nuevos métodos, obviando los antiguos y añadiendo nuevas variables de instancia (modelo) para satisfacer las necesidades de la aplicación. Las bibliotecas de clases proporcionan un método de aprendizaje ya que analizan sus definiciones de clases y los métodos de éstas.

los tipos de datos y métodos ya existentes no serán afectados por el cambio. Al modificar la representación interna de un tipo de dato ó método, podría hacerse en aquella clase de la jerarquía de clases en la que se define el tipo de datos o método. Finalmente, debido que los cambios en la estructura jerárquica de la base de datos continúan para asociar los datos que están relacionados lógicamente, el mecanismo de la herencia soporta funciones de base de datos como:

- Bloqueo
- Autorización
- Consulta

La jerarquia de clase determina una trayectoria de herencia entre la superclase y sus subclases, permitiendo al diseñador reutilizar las definiciones de clases ya existentes incorporando sus atributos y métodos.

La jerarquia de composición (generada al permitir asignar como valor de un atributo el identificador de otro objeto) es la base de la definición recursiva de un objeto complejo en términos de otros objetos.

BIBLIOTECA DE CLASES

Una clase representa un molde o patrón de un conjunto de objetos similares, el usuario define las clases a partir de las bibliotecas de clases predefinidas que son parte del DBMSOO, las bibliotecas de clases permiten a los programadores **reutilizar,** en lugar de tener que volver a crear las estructuras de datos más comúnmente necesarias, como por ejemplo: matrices, diccionarios, tablas y fechas; las bibliotecas de clase incluyen también métodos, como recorrido lineal o dispersión (Hasshing). Al igual que con un LPOO, los programadores de BDOO perfeccionan las bibliotecas de clases y las jerarquías añadiendo nuevos métodos, obviando los antiguos y añadiendo nuevas variables de instancia (modelo) para satisfacer las necesidades de la aplicación. Las bibliotecas de clases proporcionan un método de aprendizaje ya que analizan sus definiciones de clases y los métodos de éstas.

Las bibliotecas de clase de los DBMSOO difieren principalmente de las bibliotecas de clase de los LPOO en el tipo de clases que definen. Ambos incluyen clases para los tipos básicos como enteros, reales, cadenas, así como tipos agregados o totales como listas conjuntos y matrices. Las bibliotecas de clases para las BDOO incluyen clases para objetos persistentes, excepciones, directorios, bloqueos, esquemas, y demás funcionalidad propia de una base de datos. La siguiente figura muestra parte de una típica biblioteca de clase para una BDOO y la biblioteca de clase para un LPOO.

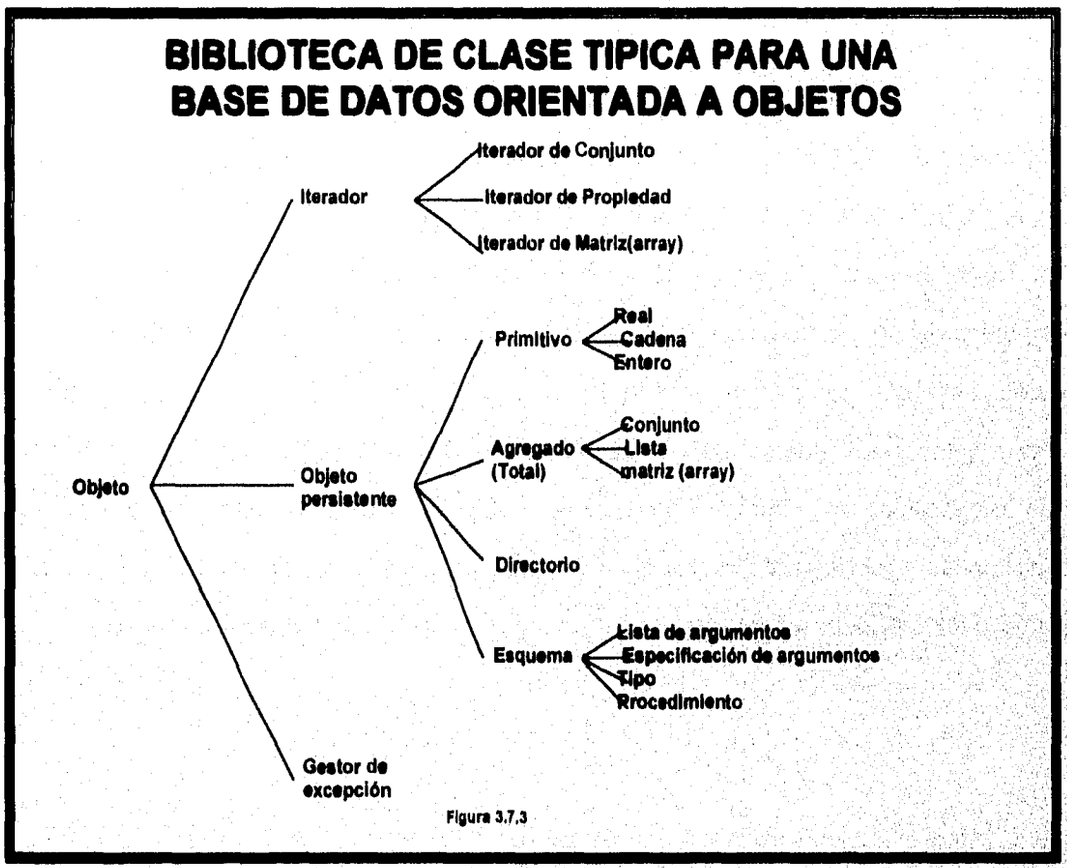


Figura 3.7.3

CLASES DE COLECCIONES PREDEFINIDAS

Las clases de colecciones es una super clase que permite la definición de estructuras simples de objetos en grupos organizadas de cierta forma: El conjunto más común de estas clases es:

- **Arreglos**
- **Cadenas**
- **Diccionarios**
- **Bolsas**
- **Conjuntos**

Las cuales pueden hacer las siguientes operaciones :

Buscar, agregar, acceder y cambiar los elementos de la colección.

Alteración de los elementos de la colección por el mensaje **do:** para procesar todos los elementos de una colección.

3.8 TECNICAS PARA CONSTRUIR UN DBMSOO

Ahora que los DBMSOO surgen y se comercializan, es tiempo de profundizar un poco y mirar dentro de la tecnología y considerar como son construidos actualmente los DBMSOOs. El primero y el más obvio descubrimiento es al igual que los lenguajes de programación. Los DBMSOOs también existen en dos formas básicas: puros e híbridos; los puros son construidos con la premisa de manejar y almacenar sólo objetos, se fundamentan en el modelo de datos OO; los DBMSOOs Híbridos son construidos a partir de los mejores DBMSR con todas sus características relacionales y adicionando el manejo de objetos simples, a continuación se examinan primero los híbridos, o también conocidos como DBMS Relacionales extendidos.

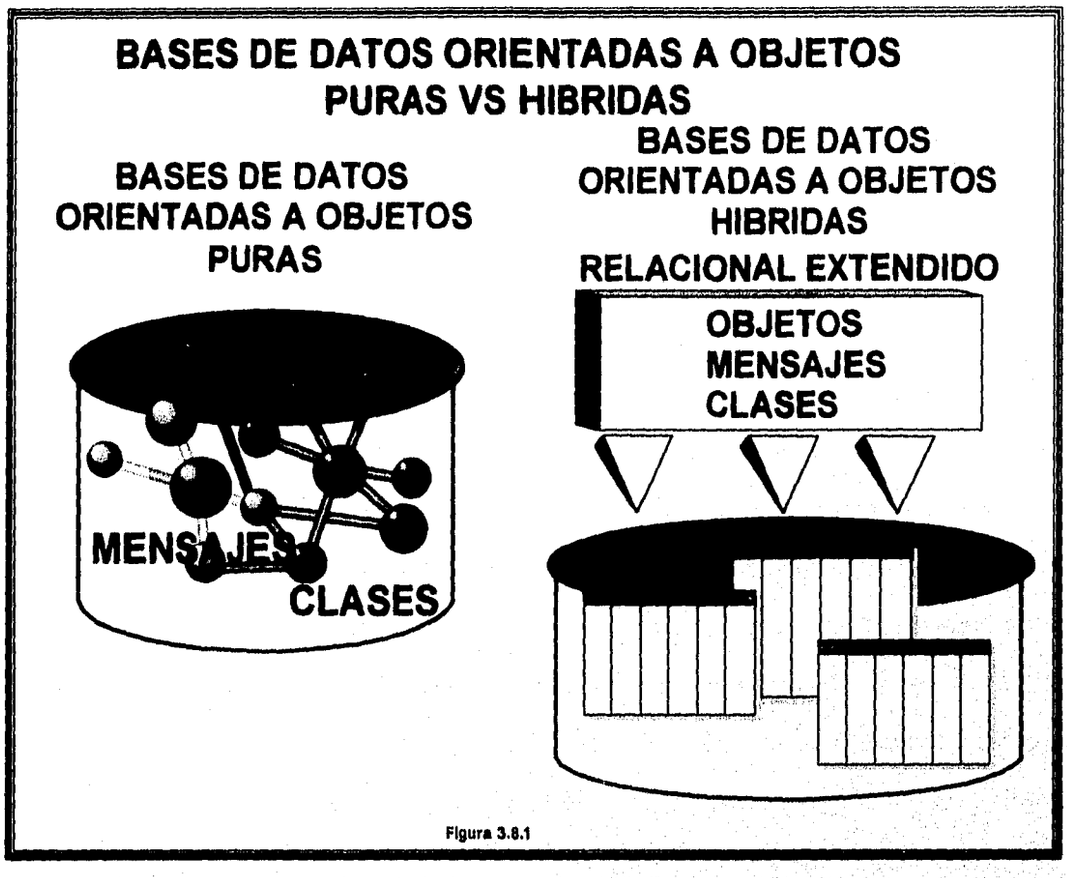


Figura 3.8.1

USANDO OBJETOS CON BASES DE DATOS RELACIONALES

La mezcla de objetos con bases de datos relacionales debe verse con esceptisismo; la razón es que las bases de datos orientadas a objetos hacen mucho más de todo lo que las bases de datos relacionales hacen, por consiguiente si se necesita el poderío de una base de datos orientada a objetos, no tiene caso usar también bases de datos relacionales (a menos que se halla efectuado un parche, que usa objetos, a un programa anterior que estaba accesando una base de datos relacional). En general si se está satisfecho con una base de datos relacional, debido a que la estructura y los datos no son complejos, no hay razón para pasarnos a una base de datos orientada a objetos, con su nomenclatura y semántica nueva y con su inmadurez (por ser productos recientes). Por otro lado si se emplearan sólo unos cuantos objetos en memoria no hay razón para guardarlos en disco (en forma permanente), bastará con crearlos de nuevo cada vez que el programa se ejecute, es decir, estos objetos sólo

son residentes en memoria primaria, es lo que ofrece C++ "Objetos en memoria solamente", si se desea hacerlo permanente entonces se utiliza una base de datos orientada a objetos.

El manejo de Objetos puede ser incorporado a los sistemas tradicionales ya sea Jerárquico, Red o Relacional, el mejor ejemplo: ACCESS FOR WINDOWS, FOX PRO FOR WINDOWS, PARADOX FOR WINDOWS, INFORMIX Nueva Era y ORACLE que incorpora dentro de sus tipos de datos un campo que adiciona extensiones de objetos a la base de datos. Es obvio que todos los esfuerzos por agregar objetos están basados en el modelo relacional. No obstante me parece inverosímil que sea factible que los modelos Jerárquicos y de Red puedan ser nuevamente utilizados en estos tiempos gracias a la Tecnología OO. Existen dos métodos principales para que estos DBMS Relacionales extendidos soporten objetos, uno de ellos es el manejo de BLOBS, es decir permite que la base de datos contenga apuntadores hacia los objetos, como se muestra en la siguiente figura:

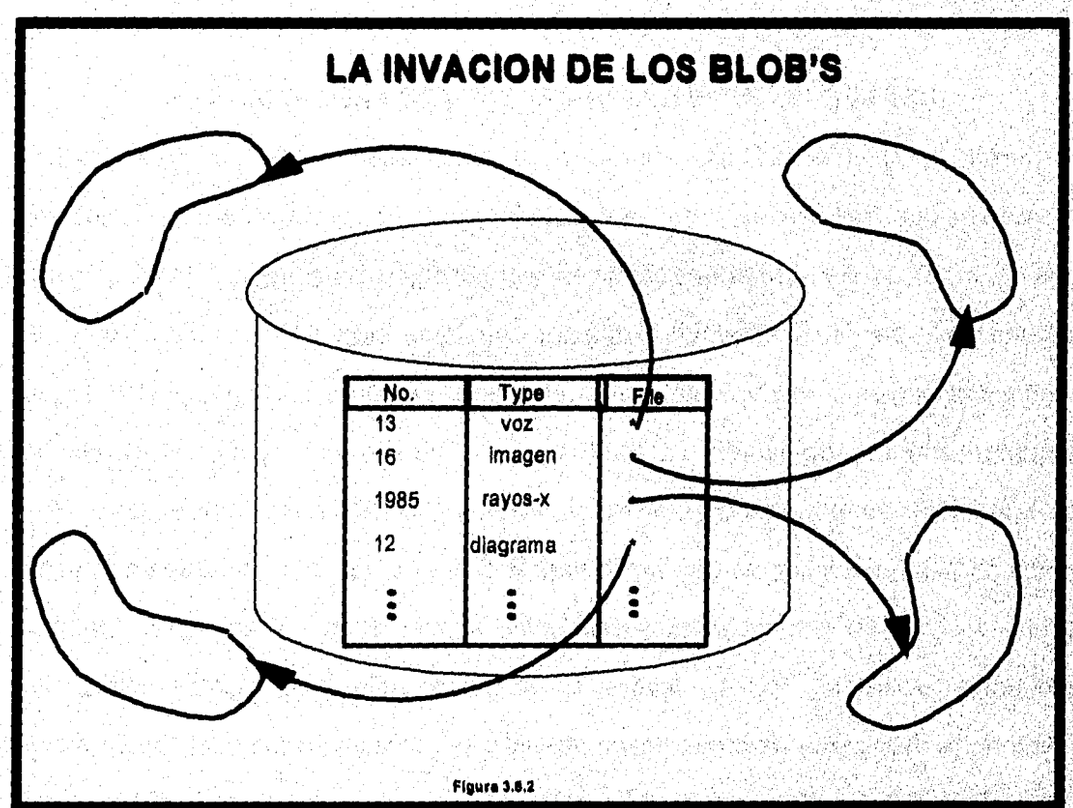


Figura 3.8.2

con lo cual los objetos son otro tipo de datos en la tabla, los operadores de manipulación de datos funcionan a nivel tabla y los objetos se llaman en un entorno externo a la base de datos en el cual los mensajes pueden actuar sobre ellos. Este método limita las clases de consulta que pueden realizarse acerca de ellos, pero evita la sobre carga de reducir el objeto en tablas para su almacenamiento, como lo hace el siguiente método.

BLOBS

Uno de los mecanismos para realizar esta extensión es conocida como BLOB, un acrónimo para un **B**inary **L**arge **Ob**ject. Los BLOBs juegan un papel determinante en el modelo relacional extendido, ya que básicamente los BLOBs son archivos que contienen información binaria representando una imagen, un procedimiento, una estructura compleja o cualquier cosa, en una base de datos relacional, la base de datos contiene referencias a esos archivos, así que eso hace manejable la información que contiene, albeit indirectly.

El uso de BLOBS no es una solución elegante al problema de almacenar información de multimedia. Almacenar esta información en archivos externos niega muchas de las protecciones naturales que brinda un DBMS. Lo más importante es que los BLOBS no puede contener otro BLOB, así que ellos no pueden tomar la forma de objetos compuestos. También BLOBS ignora generalmente el aspecto de objetos.

El segundo método crea una concha (shell) que permite a los objetos actuar como tablas, la concha (shell) descompone las estructuras de objeto en archivos planos para que puedan ser almacenadas en tablas relacionales como indica la siguiente figura:

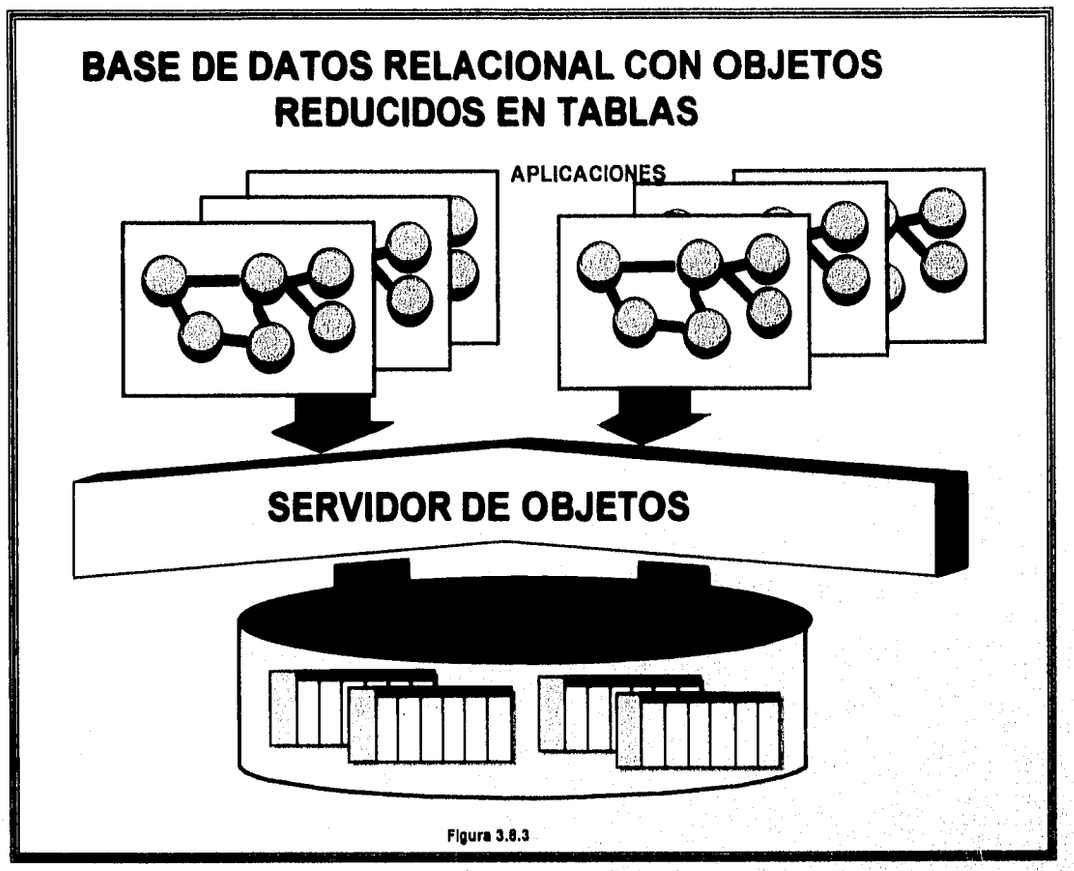


Figura 3.8.3

Los objetos actúan entonces como tablas virtuales y se les llama cuando se les necesita, utilizando un lenguaje estandar de consulta como el SQL. Estas conchas (shells) almacenan información sobre los datos como objetos, pero los datos en sí permanecen en tablas relacionales. La concha almacena y recupera solamente objetos enteros, por lo que no soporta objetos complejos, por ello el método es adecuado cuando los objetos pueden indexarse por campos normales de datos. Este método de objeto extendido sufre también un inconveniente en el rendimiento. El proceso relacional busca hacer coincidir, es más lento que la búsqueda en una base de datos Orientada a Objetos pura, donde las consultas tienen referencias directas a los datos y se soporta objetos complejos.

La siguiente figura muestra cómo se almacenan instancias de un objeto en registros de una BDR extendida.

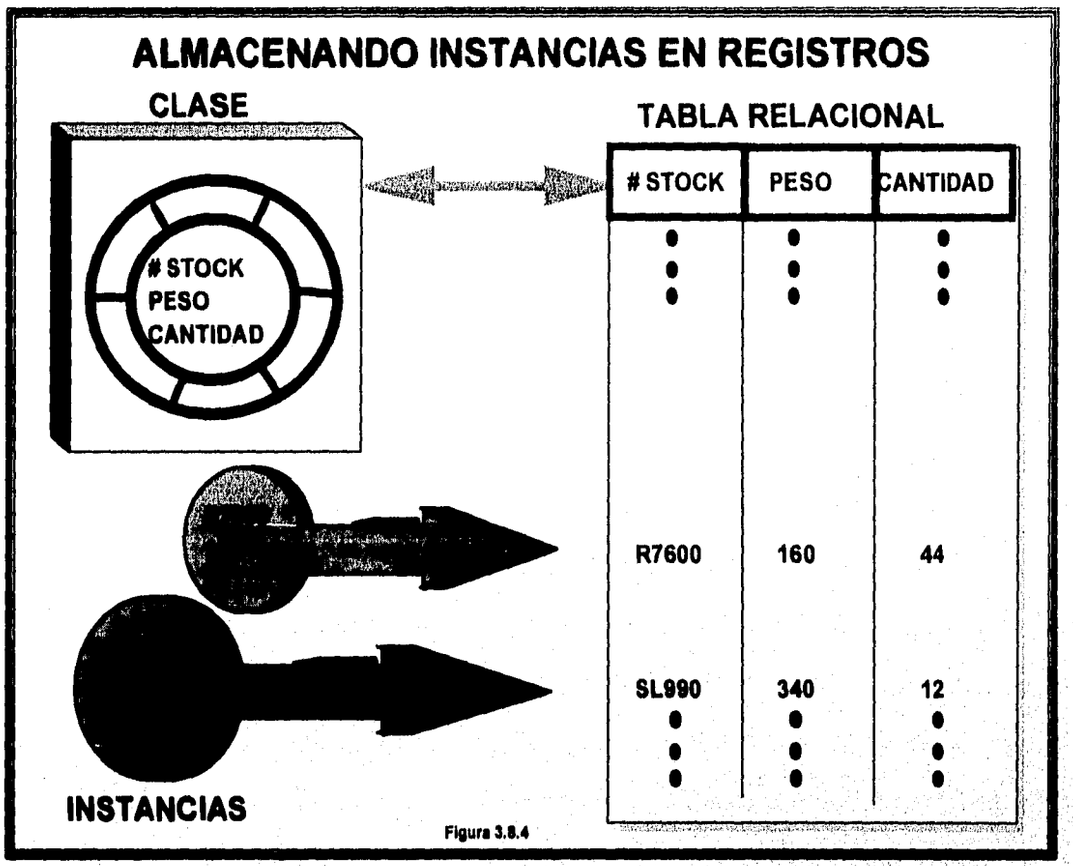


Figura 3.8.4

Esta tecnología es equivalente a la de los lenguajes de programación orientados a objetos híbridos. Los vendedores de DBMS relacionales vienen estableciendo que la solución al problema del manejo de objetos se resuelve mediante el uso de sistemas relacionales que ellos llaman Modelo Relacional Extendido, los cuales contienen como base toda la potencia de la tecnología relacional.

Sin embargo, existen algunos problemas al emplear el modelo relacional extendido, estas dificultades son las siguientes:

- Solo trabnajancon lenguajes de programacion bien definidos. Los DBMS Relacionales-Extendidos requieren que todos los valores de una columna sean del mismo tipo de dato y longitud fijos. Las variables de C++ conocen este criterio, pero Smalltalk juega haciendo estragos con este requerimiento.

- No existe Facilidad para que el usuario defina sus propios tipos de datos. La información almacenada en los sistema relacionales restringe los tipos de datos a un conjunto de tipos de datos ya definidos por él. La habilidad para definir nuevos tipos de datos en forma de clases es un aspecto clave de la tecnología OO.
- Los DBMS Relacionales no están preparados para almacenar procedimientos complejos. Esto es un problema esencial ya que la Tecnología de OO encapsula los datos con los métodos (procedimientos).
- Los DBMS Relacionales son malos para el acceso a través de la navegación. La tecnología de Objetos depende excesivamente de los Identificadores únicos de Objetos Ids, que son apuntadores a los Objetos, mientras que el modelo relacional evita los Ids ya que sus accesos están diseñados en base a su asociatividad, es decir, las relaciones están en función de los valores de las columnas de las tablas más no de las columnas de la tabla en sí.

Todos estos problemas empiezan al tratar de incorporar dos modelos de datos diferentes como son el Relacional y el orientado a Objetos. En un sistema relacional extenso, los datos pueden ser almacenados y los vendedores de estos productos van dirigiendo la solución de estos problemas a lo que llaman ellos el modelo relacional extendido, (Cattell, Object Data Managament: Adison Wesley, 1991). En un sistema relacional extendido un tipo de datos puede estar almacenado y los usuarios pueden definir nuevos tipos de datos, los procedimientos pueden ser almacenados directamente en la base de datos o como referencias en archivos separados.

Los DBMSOO Puros

Así como los lenguajes de programación Orientados a Objetos puros, los DBMSOOs Puros son diseñados y construidos con el único propósito de manejar objetos. En general un DBMSOOs de este tipo son vendidos por compañías que se especializan en esta nueva tecnología de DBMS. Esta tesis se enfoca a este tipo de DBMSOO y se mencionan productos existentes en el mercado, en el capitulo IV.

Los vendedores de DBMSOO Puros tienen la ventaja de estar a la vanguardia de esta filosofía y tecnología. Por empezar desde la línea de partida, ellos pueden crear productos que son solicitados precisamente para los requerimientos de manejo de los objetos aunque también toman ventajas de los avances de las tecnología, como Archivos Planos, Relacional, Jerárquica y de Red. Por ejemplo algunos DBMSOO fueron diseñados desde el principio para ofrecer almacenamiento de información distribuida. En contraste esta es una característica que inició lentamente al ser incorporada a los sistemas relacionales.

DBMSR Extendidos a Objetos VS DBMSOO Puros

Las extensiones orientadas a objetos son aceptadas por proveedores como INFORMIX, SYBASE, etc. los cuales ya han lanzado sus bases de datos relacionales de segunda generación (extendidas) cuyos productos proporcionan reglas almacenadas junto con datos en forma de disparadores y procedimientos almacenados. Otro ejemplo de un modelo relacional que ha sido extendido para soportar tipos de datos no convencionales, es Object Management Intelligent DataBase INGRES Corporation. Ambas bases de datos aunque más ricas, son todavía relacionales, con su estructura determinada fundamentalmente por tablas unidas.

Eventualmente las bases de datos orientadas a objetos puras están comercialmente disponibles en los Estados Unidos de América y Europa, aún no han sido promovidas en México. Estas bases de datos almacenan y recuperan estructuras con información interelacionada y compleja, las cuales actúan como depósitos para conjuntos de objetos interrelacionados, que se componen de datos y métodos, además de soportar la riqueza de las aplicaciones (multimedia hipermedia, etc.) que hoy no puede soportar el modelo relacional extendido, las bases de datos orientadas a objetos puras, también soportan sistemas desarrollados con lenguajes de programación orientados a objetos.

- Los DBMSOO Puros proveén un conjunto de herramientas para el manejo eficiente de objetos, mensajes y clases, los mecanismos básicos de la tecnología OO.
- Los DBMSOO Puros son mejores en cuanto al uso de navegación mediante punteros, lo cual hace el acceso a los objetos de una forma rápida y eficiente al trabajar con estructuras complejas de información.
- Los Sistemas relacionales extendidos son mejores en cuanto a accesos asociativos y son generalmente más rápidos en el proceso de transacción con estructuras de datos simples.
- Los DBMS Relacionales Extendidos hasta ahora proveén mejores herramientas para definir bases de datos y construir aplicaciones.

Estos son unos cuantos de los puntos para ser considerados al escoger entre un DBMSOO y un DBMSR Extendido a OO.

Estas son otras consideraciones, pero puede ser prematuro el discutirlas porque la tecnología va cambiando rápidamente.

Aunque los DBMSR del mercado se acoplan al modelo relacional extendido son primeramente teóricos hasta ahora, así que es difícil, comentar sus méritos en cualquier detalle. Similarmente los DBMSOO Puros están madurando tan rápido que algunas de sus limitaciones están siendo eliminadas, adelantándose a las complicadas comparaciones entre estas dos tecnologías que convergen rápidamente. Los DBMSOO puros ofrecen mejorar cada vez sus mecanismos que soportan eficientemente el acceso asociativo y estas herramientas se enriquecen con el tiempo de todas las demás tecnologías, por otro lado los DBMS relacionales extendidos van incrementando las capacidades para el manejo de procedimientos (métodos) y herencia entre otras características de la orientación a objetos. Los DBMS relacionales extendidos son realmente principiantes que se encuentran con dificultades para agregar Ids a los objetos en sus tablas y introducir punteros de navegación como alternativas para el acceso asociativo.

Para los vendedores de DBMS, la diferencia entre un DBMSOO y un DBMS Relacional Extendido pueden converger en un punto indistinguible, el simple hecho de agregar al modelo relacional extendido punteros para el acceso por medio de la navegación, nos trae la pregunta si esta navegación está fundada en el álgebra relacional ?. Con el tiempo los vendedores fueron distinguiendo entre el DBMSOO puro y el DBMSR extendido con el objeto de negociar las base de datos.

La convergencia entre los DBMSOO Puros y los Híbridos (DBMSR Extendidos) se aproxima tomando años para complementarse, mientras tanto, la selección de la tecnología deberá tener el siguiente criterio:

Si su necesidad actual es el proceso de transacciones y el acceso a los datos mediante SQL, entonces el DBMS Relacional extendido es su elección, ya que ofrece un camino para el manejo de objetos en forma limitada pero quizá de acuerdo a sus necesidades. Por otro lado si su necesidad es el poder que representa la tecnología de orientación a objetos, con la rapidez en el manejo de estructuras complejas, indudablemente su elección debe ser un DBMSOO Puro.

Los sistemas de información requieren totalmente el poder de cada componente de la tecnología orientada a objetos, por esta razón lo que resta de este capítulo estará enfocado a los DBMSOO Puros. Este enfoque no estará dirigido a los DBMS Relacionales extendidos, pero refleja las realidades actuales. En el futuro si un vendedor le ofrece un DBMS Relacional extendido que tenga la flexibilidad y funcionamiento de un DBMSOO Puro, usted deberá considerar seriamente cada componente de ese producto que será el centro de su sistema de base de datos basado en la orientación a objetos.

3.9 COMPONENTES DE UN OODBMS

Un OODBMS se define como un DBMS que soporta directamente un modelo basado en el paradigma orientado a objetos y los tres componentes básicos se muestran en la siguiente figura:

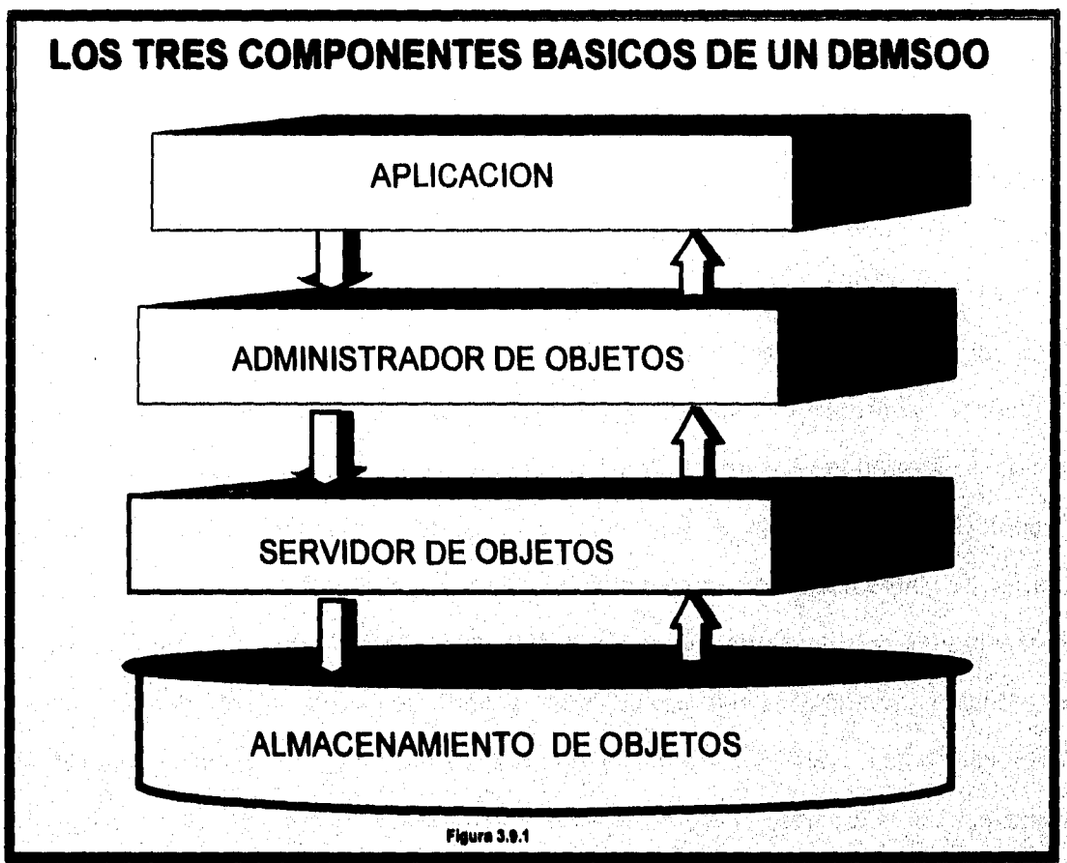


Figura 3.9.1

Cada vendedor de DBMSOO aborda diferentes enfoques para construir su producto. Cada producto es estructurado de forma única pero como quiera que sea, todos tienen algo en común, una sobre vista de la arquitectura común ayudará a entender cómo trabajan los DBMSOO. Esto también ayudará a conocer los puntos básicos que ofrece cada vendedor de DBMSOO Puro

Esencialmente hay tres componentes básicos en un DBMSOO Puro: **el manejador de objetos, el servidor de objetos y el almacenador de objetos**, como se muestra en la figura anterior. Los componentes son mostrados por bloques aproximadamente en el

siguiente orden: La aplicación interactúa con el administrador de objetos que es el encargado de administrar los recursos y el cual colabora intensamente con el servidor de objetos que aprovecha el acceso del almacenador de objetos

Hay muchas formas de combinar estos componentes y la razón de que estos componentes estén separados es que ellos pueden ser mezclados y ensamblados en una extensa variedad de convinaciones con el fin de optimizar la manipulación de los objetos, la ilustración anterior muestra un caso simple en donde cada aplicación de usuario requiere solamente uno de cada componente, después, las partes de varios componentes tienen bien definidos este contexto. Ahora examinemos más sofisticadas formas de combinar estos módulos.

El Administrador de Objetos.

El papel básico de los administradores de objetos es el administrar la memoria caché local de los objetos en una aplicación individual. Esencialmente las aplicaciones checan fuera de la base de datos a los objetos, usando los servicios del servidor de objetos que es el encargado de ganar acceso para almacenar a los objetos, éstos checan los objetos y los regresan cuando ellos han terminado de ser utilizados. Típicamente la memoria caché local es implementada con memoria virtual para evitar las limitaciones de tamaño, particionando el disco si es necesario para traer el objeto actual. El caché local de objetos actúa en un espacio de trabajo temporal para distribuir los objetos afuera del entorno de la base de datos. Es una regla que todos los nuevos objetos sean creados primero en caché y posteriormente la base de datos se encarga de almacenarlo. Esta misma regla es aplicable en las modificaciones de los objetos.

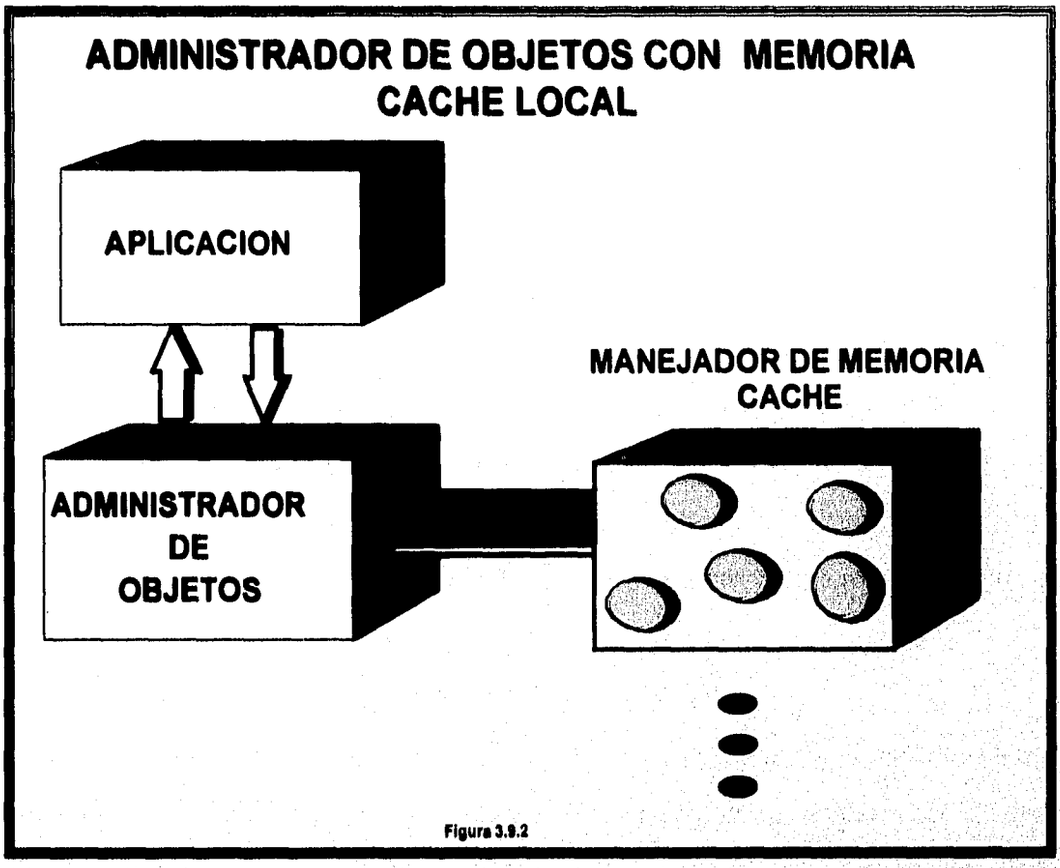


Figura 3.9.2

El administrador de objetos trabaja también con el servidor de objetos proporcionándole otros tipos de servicios, éstos incluyen el manejo de los procesos de entrada/salida, encargándose de generar nuevos objetos o modificando los ya existentes en la base de datos, y ejecutando las transacciones necesarias entre los formatos, el programa orientado a objetos y la base de datos OO. Un ejemplo de éste último servicio podría convertir un programa OO de C++ désde un programa con formato de Opal, haciendo esta función internamente el administrador de objetos del DBMSOO (GemStoned).

El Servidor de Objetos

El papel que desempeña el servidor de objetos es manejar y separar, la caché que comparten los objetos. Esta caché es implementada en memoria distribuida, así que

puede accesar muchas aplicaciones simultáneamente. Esencialmente el Servidor de Objetos usa esta caché para coordinar el acceso del almacenamiento de los objetos.

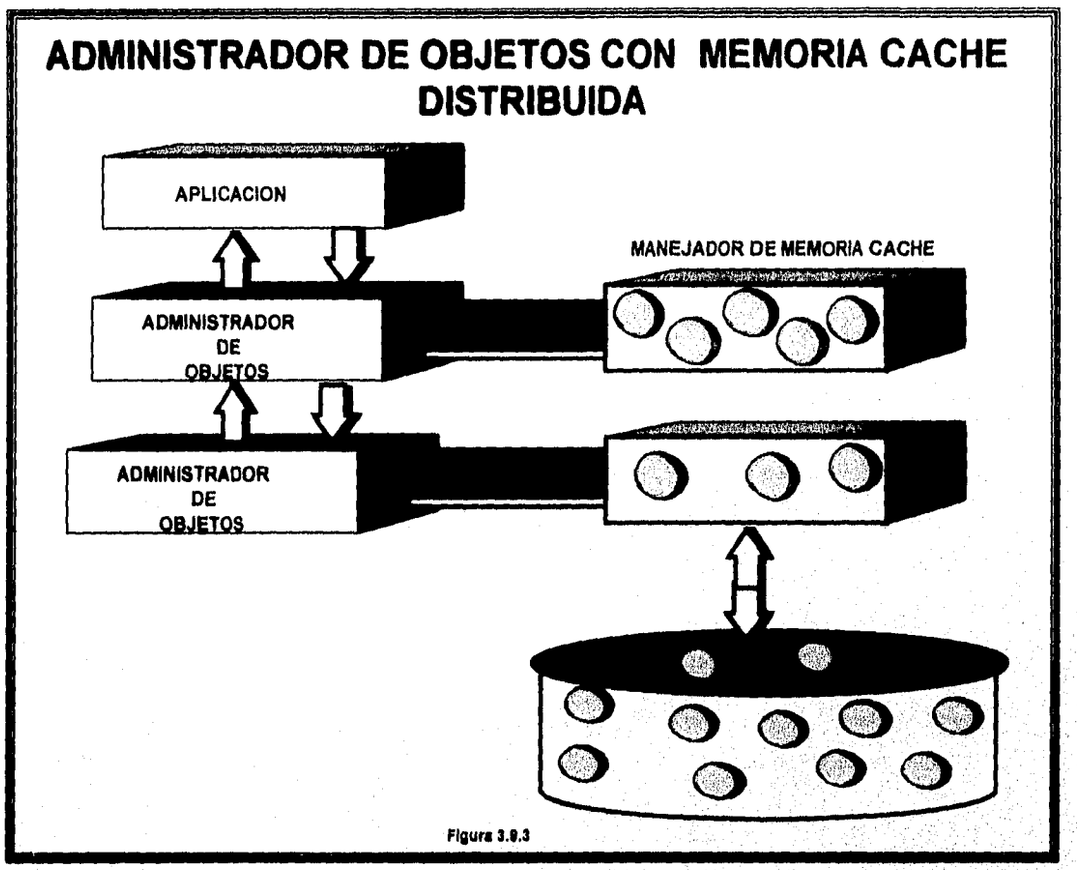


Figura 3.0.3

Por ejemplo, observando todas las operaciones que son realizadas por el servidor de objetos, usando su memoria caché distribuida, se puede observar cómo son evitados los conflictos con las diferentes aplicaciones que tratan de accesar al mismo objeto. Observe cómo la administración de transacciones es una de la principales responsabilidades del servidor de objetos. El servidor de objetos también se encarga de manipular los diferentes formatos físicos de los datos entre varias máquinas con diferentes Servers. Si un administrador de objetos está corriendo en un Sun 4 y el Server está ejecutándose en un IBM RS/6000, el servidor de objetos automáticamente trasladará los datos (objetos) almacenados con el formato de la Sun, leyendo así los objetos en forma correcta. Esta facilidad es una característica

de la mayoría de los DBMSOO Puros que se ejecutan de forma distribuida a través de varias plataformas heterogéneas de hardware.

Combinando Administradores y Servidores de Objetos

Los administradores y servidores de objetos pueden ser combinados de diversas formas. Los mejores DBMSOO permiten realizar verdaderamente sofisticadas combinaciones entre el Administrador, Servidor y el Almacenador de Objetos. En adición ellos permiten un amplio rango de configuraciones semejantes entre varios de sus componentes que les permiten correr en forma semejante en diferentes máquinas, de tal forma que se optimizan los recursos computacionales.

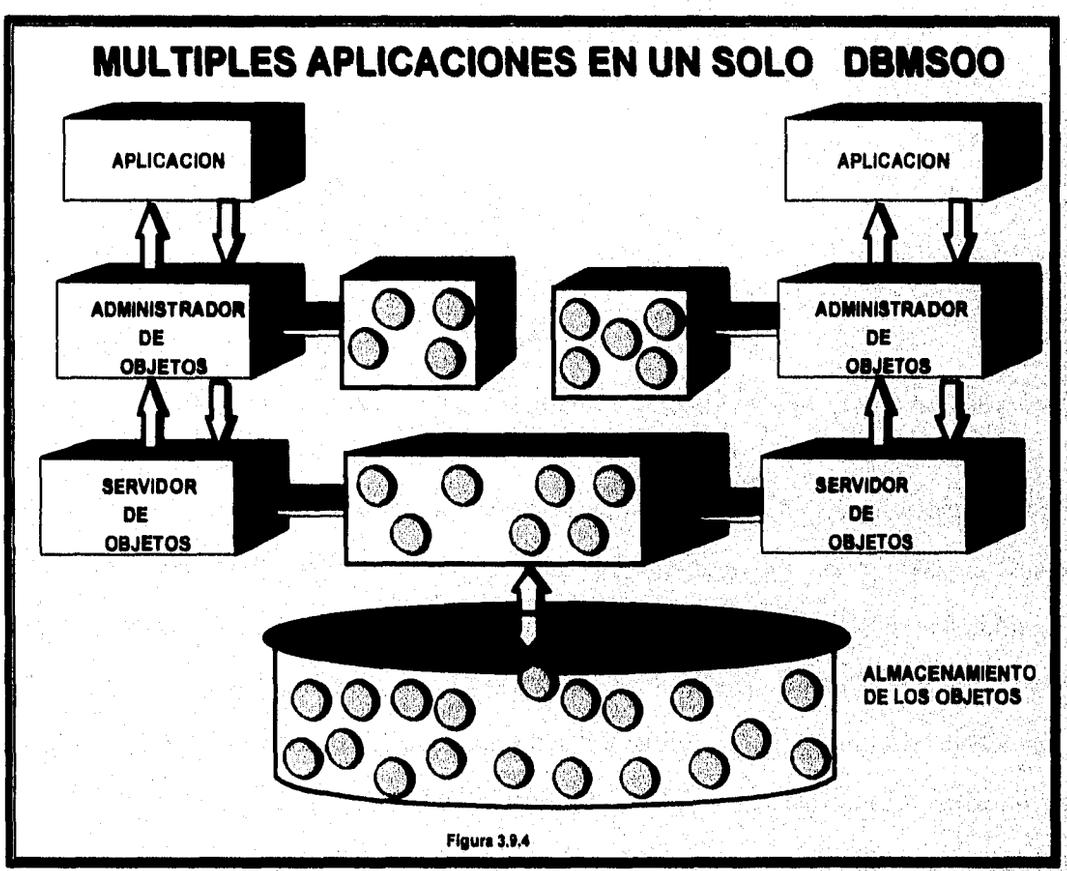


Figura 3.9.4

En una configuración común, diferentes aplicaciones pueden tener acceso a una sencilla base de datos. En este caso cada aplicación tiene sus propios

de la mayoría de los DBMSOO Puros que se ejecutan de forma distribuida a través de varias plataformas heterogéneas de hardware.

Combinando Administradores y Servidores de Objetos

Los administradores y servidores de objetos pueden ser combinados de diversas formas. Los mejores DBMSOO permiten realizar verdaderamente sofisticadas combinaciones entre el Administrador, Servidor y el Almacenador de Objetos. En adición ellos permiten un amplio rango de configuraciones semejantes entre varios de sus componentes que les permiten correr en forma semejante en diferentes máquinas, de tal forma que se optimizan los recursos computacionales.

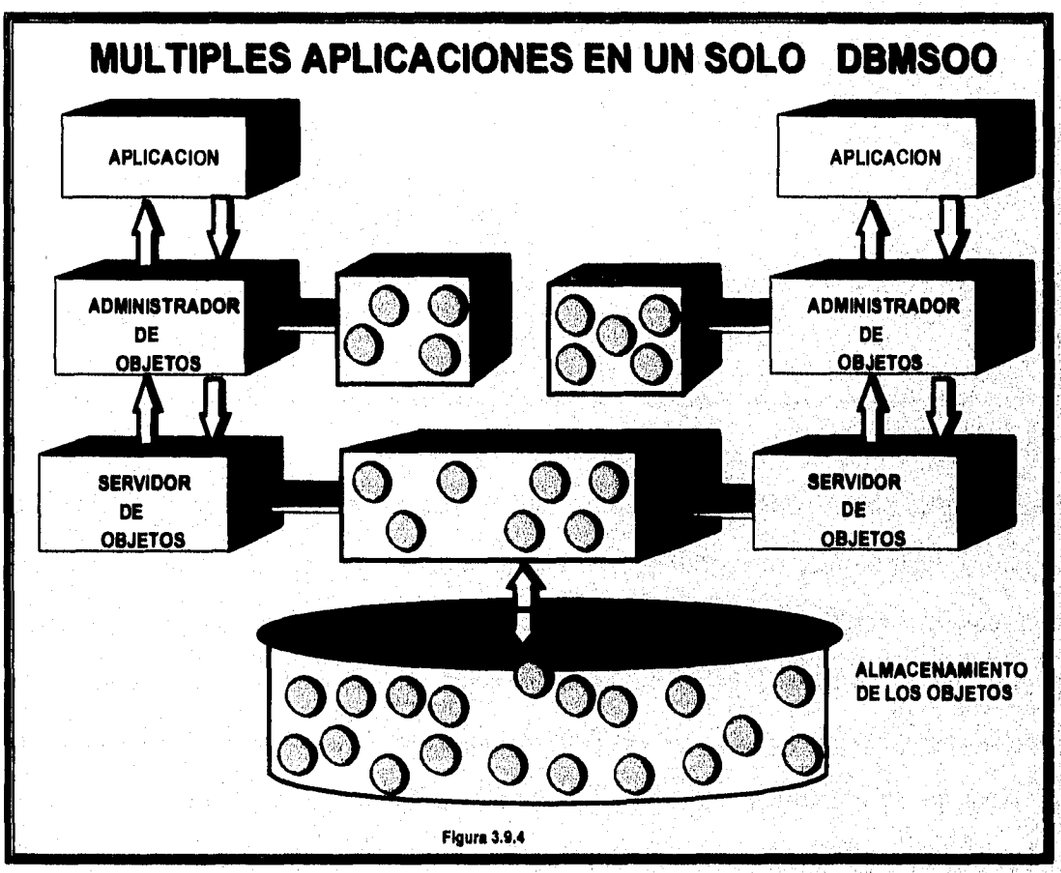


Figura 3.9.4

En una configuración común, diferentes aplicaciones pueden tener acceso a una sencilla base de datos. En este caso cada aplicación tiene sus propios

administradores y servidores de objetos. Pero varios servidores vienen interactuando con el almacenador de objetos por medio de la memoria caché distribuida mantenida por el servidor de objetos.

Por otro lado una aplicación puede accesar a múltiples bases de datos, que utilizan por separado el administrador y el servidor de objetos. Esta es actualmente una forma limitante de administrar la distribución de los objetos.

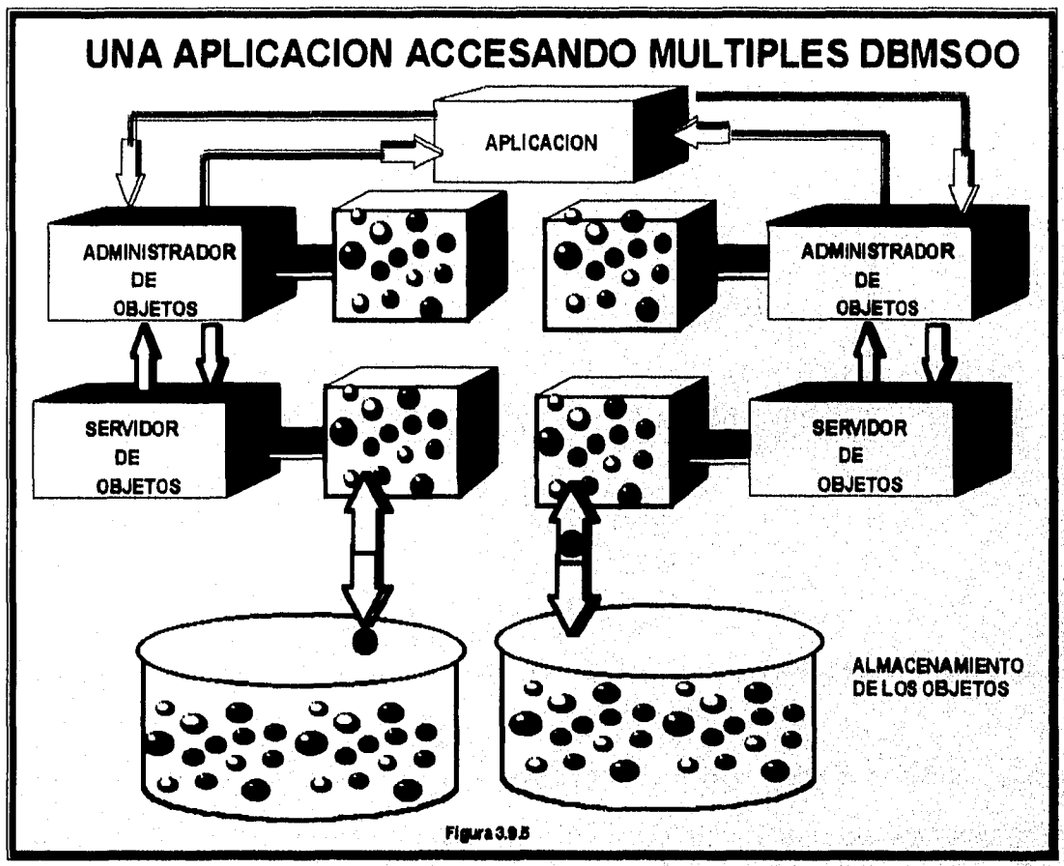


Figura 3.9.5

Enlazadores VS Llamadas a Procedimientos Remotos

De cualquier forma, varios administradores y servidores de objetos pueden verse envueltos en un sistema, estas son varias opciones con las cuales pueden actuar de forma conjunta. La simple acción de enlazar una aplicación, implica que el administrador de objetos y el servidor de objetos realicen un proceso sencillo empleando para ello una técnica estándar de un programa enlazador (linking). Esta

opción proveé de máxima velocidad, ya que varios componentes interactúan aproximándose en las llamadas a funciones. Esta configuración es usada típicamente en plataformas mono-usuario, tal como en CAD y otras aplicaciones de Ingeniería.

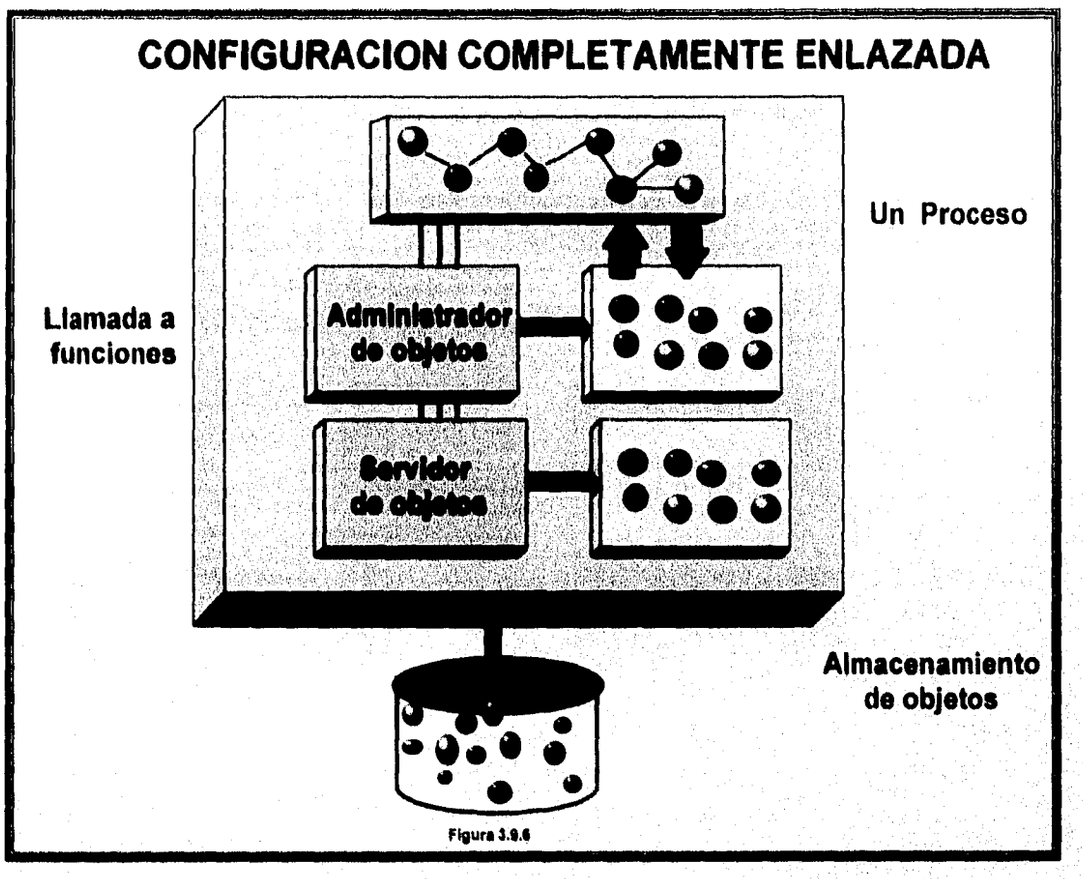


Figura 3.9.6

Otras opciones vienen sobre el movimiento de varios componentes en otros procesos, otros en una misma o en diferentes máquinas. Estos procesos de comunicación se realizan por medio de llamadas a procedimientos remotos (RPC Remote Procedure Calls), los cuales son más lentos que las llamadas directas a las funciones, pero ofrecen más flexibilidad en la localización de recursos computacionales.

Pensemos en un extremo lógico, en una aplicación, el administrador de objeto está interactuando con el servidor de objetos y el almacenador de objetos puede correr en diferentes máquinas. Esta clase de distribución es muy común en aplicaciones

opción proveé de máxima velocidad, ya que varios componentes interactúan aproximándose en las llamadas a funciones. Esta configuración es usada típicamente en plataformas mono-usuario, tal como en CAD y otras aplicaciones de Ingeniería.

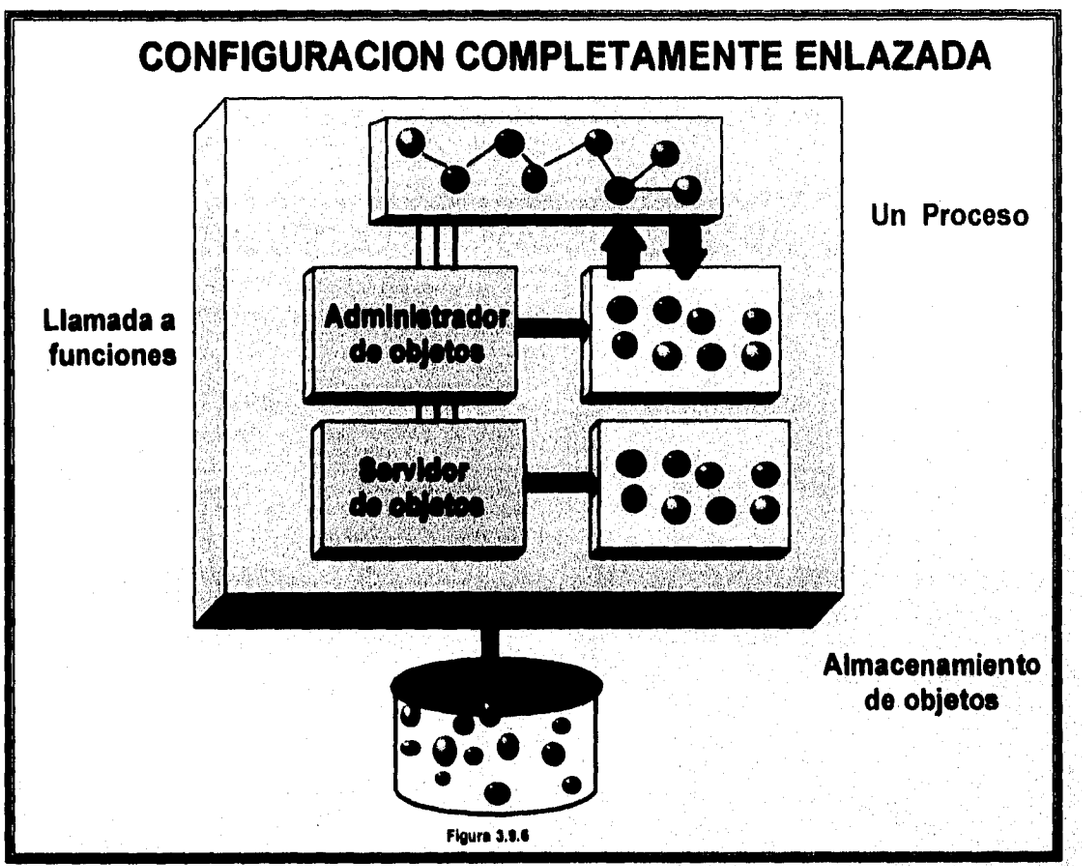


Figura 3.9.6

Otras opciones vienen sobre el movimiento de varios componentes en otros procesos, otros en una misma o en diferentes máquinas. Estos procesos de comunicación se realizan por medio de llamadas a procedimientos remotos (RPC Remote Procedure Calls), los cuales son más lentos que las llamadas directas a las funciones, pero ofrecen más flexibilidad en la localización de recursos computacionales.

Pensemos en un extremo lógico, en una aplicación, el administrador de objeto está interactuando con el servidor de objetos y el almacenador de objetos puede correr en diferentes máquinas. Esta clase de distribución es muy común en aplicaciones

multiusuario que necesitan optimizar los recursos computacionales con los que cuentan. Algunos productos de bases de datos permiten una buena comunicación entre estos componentes que emplean simultáneamente diferentes protocolos en una red.

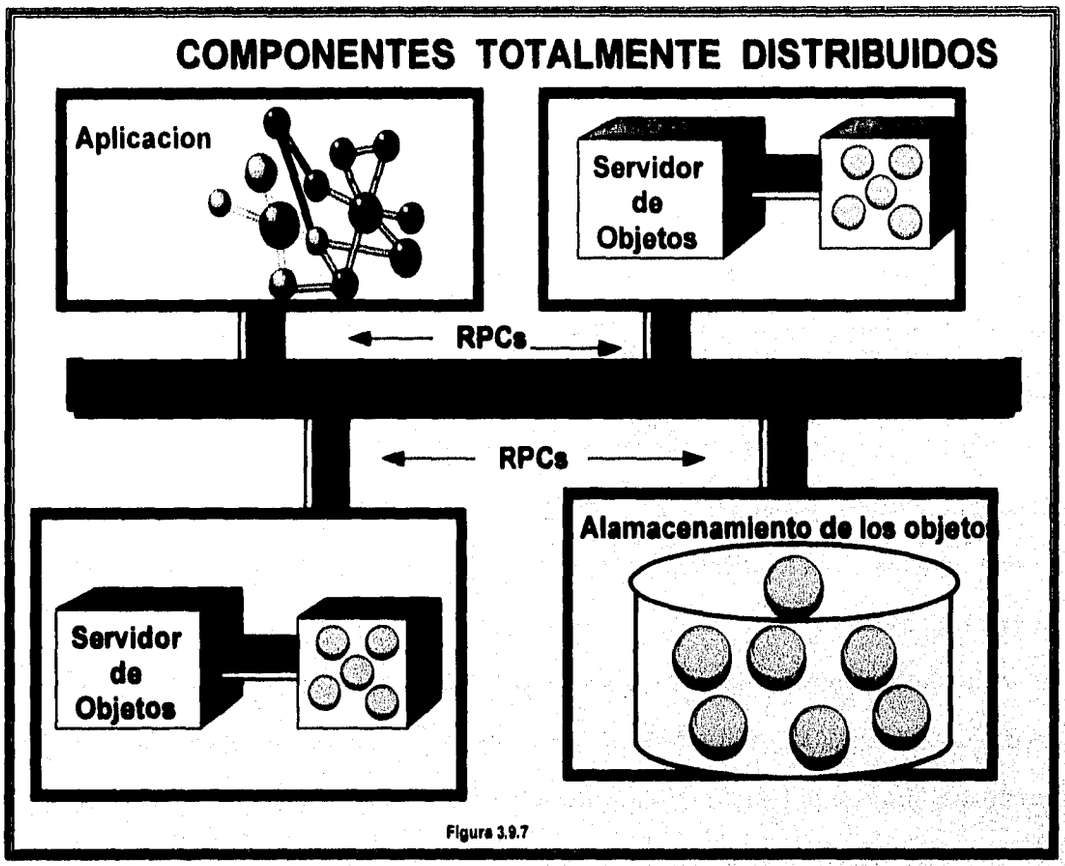


Figura 3.9.7

3.10 ENTENDIENDO EL DISEÑO DE UN DBMSOO

No hay otro camino más directo para construir un DBMSOO que diseñándolo y construyéndolo con técnicas orientadas a objetos, además por supuesto, de construirlo con un LPOO. Un DBMSOO es un producto de software construido con una serie de **tradeoffs** de numerosas dimensiones, la combinación de estas decisiones definen un producto único. En este capítulo consideraremos algunas decisiones importantes y sus posibles consecuencias.

DDL/DML Propietarios VS Extensiones del Lenguaje

Como describimos anteriormente, los primeros DBMSOO manejaban su propio diseño y construcción tratando de crear una mejor clase de DBMS que superara a los tradicionales. Estos DBMS tradicionales contienen un Lenguaje de definición de datos LDD, así como un lenguaje de manipulación de datos DML Propio; los DBMSOO no son la excepción, sin embargo, es increible que en las bases de datos primitivas, los lenguajes provéen a los DBMSOO una completa computacionalidad además, de una gran flexibilidad para realizar operaciones con los datos.

Teniendo un lenguaje propio se incrementa el poder en un DBMSOO en diferentes caminos. Primero proveé una sencilla y canónica forma de tratar con objetos persistentes prescindiendo del lenguaje en el cual éstos se definieron originalmente. Más aún, cada uno de los métodos son definidos en un lenguaje sencillo manteniéndolos en parte de la base de datos. Esto es posible en principio, aunque no hay caminos para la práctica al ejecutar los métodos directamente en el entorno de la base de datos.

En la siguiente figura se tiene separado el DDL/DML de la representación de la base de datos, por lo que existe todavía otro lenguaje que tendrán que aprender los administradores de la base de datos. Como es costumbre, se tiene conocimiento de los lenguajes de la base de datos y puede no ser visto como un problema, sino como un beneficio para algunos programadores

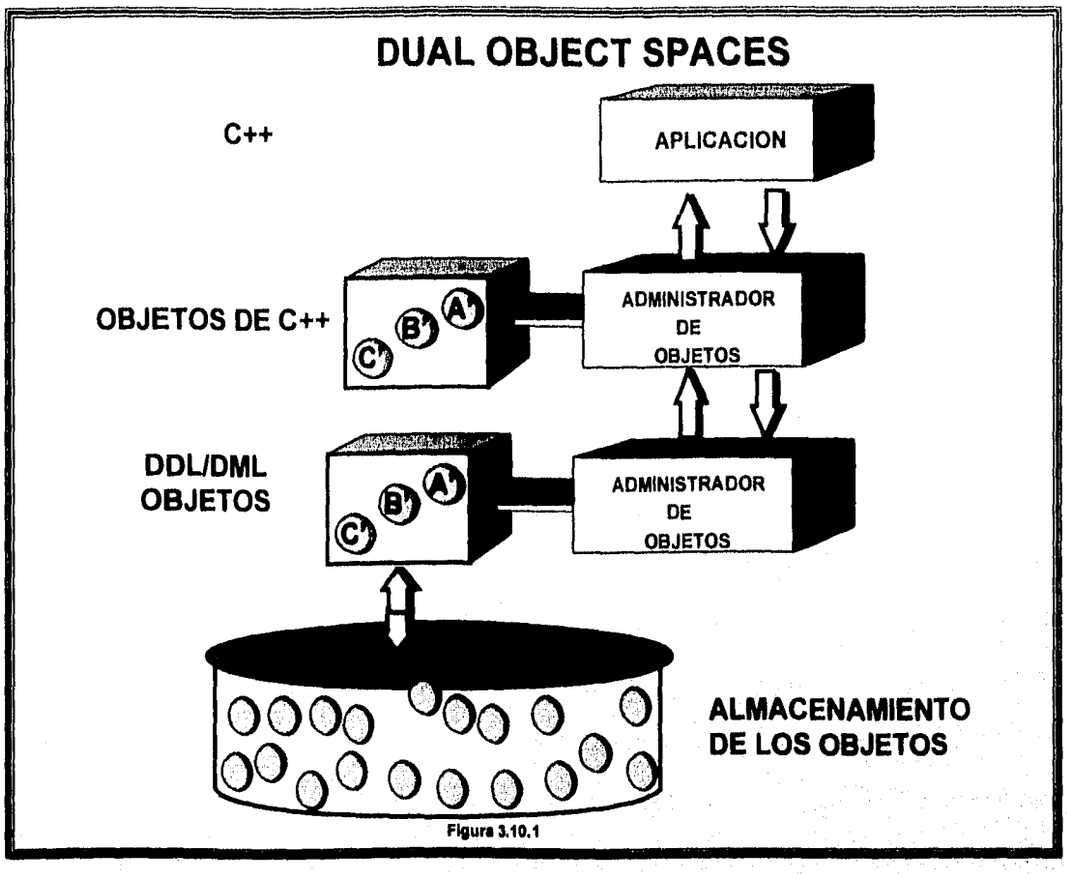


Figura 3.10.1

En conclusión, realmente el uso de dos lenguajes es un problema que incide en el espacio de los objetos, ya que al utilizar dos lenguajes en un DBMSOO todas las clases deben ser definidas dos veces, una vez en cada formato, y los cambios a estas clases deben de hacerse en ambos espacios. En contraste un DBMSOO que emplea un LPOO extendido (C++, Smalltalk, etc.), evita esta clase de problemas. Las clases son definidas sólo una vez y no requiere convertirse a otro tipo de formato de una base de datos o al formato de una aplicación. La definición del esquema de una base de datos orientada a objetos a menudo pueden ser capturados directamente desde el código fuente de una aplicación, siendo una herramienta especial en un DBMSOO, esta es una cualidad de un DBMSOO, especialmente cuando se construyen aplicaciones complejas. En algunas implementaciones, la aplicación de los métodos pueden ser persistentes y totalmente transparentes, ya sea que los objetos estén comúnmente en una aplicación o almacenados en una

base de datos. Esto puede simplificar el código de una aplicación OO, ya que los programadores no se deben preocupar por la persistencia de los objetos. No obstante la necesidad de separar el DDL/DML, remueve el nivel de protección y control tradicional de los DBMS, también trabajando con los existentes lenguajes de Objetos se puede manejar de forma transparente la persistencia de los objetos.

3.11 TECNICAS DE LOS DBMSOO PARA EL MANEJO DE LOS METODOS

Un DBMS OO maneja los métodos (procedimientos) de dos diferentes formas, algunos almacenan los métodos en archivos externos a la base de datos, y otros DBMSOO almacenan sus métodos junto con los datos en la base de datos. La primera técnica es propia de los DBMS que utilizan extensiones de lenguaje (por lo general C++, smalttalk) y la segunda opción es más común en los DBMSOO que tienen un DDL/DML propietario. La primera estrategia es semejante al almacenamiento de las librerías de los DBMS tradicionales en donde cada programa de aplicación interactúa con el DBMS para enlazar subrutinas especiales contenidas en los archivos externos al DBMS. (Con la OO los procedimientos se manejan en forma más sencilla).

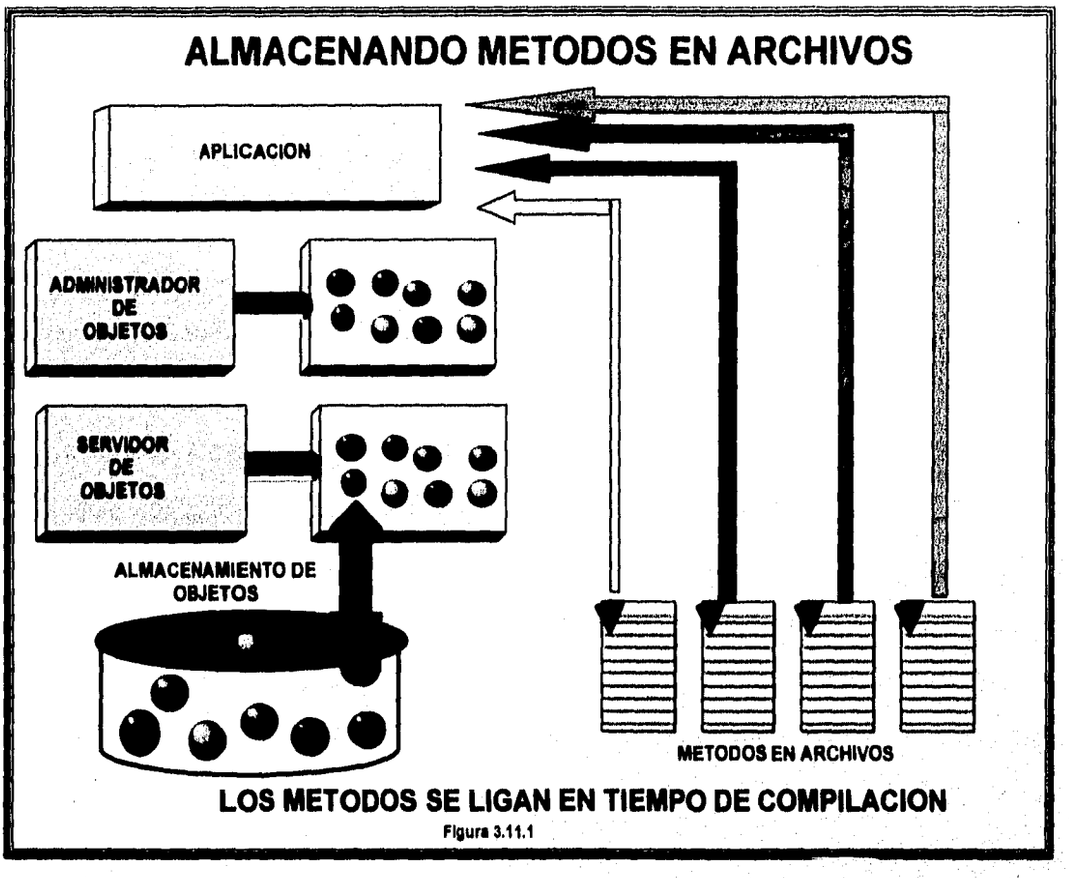


Figura 3.11.1

Sin embargo, uno de los principales problemas de este tipo de acceso tradicional para accesar a las librerías, es que hay que buscar el procedimiento correcto para accesar y tomar el método correcto, la OO relaciona los datos y los procedimientos dentro de una cápsula, por lo que si usted conoce el dato, luego entonces inmediatamente el método está identificado.

Con el acceso externo de la BD a las librerías que contienen los métodos, la modificación no es automática ya que las modificaciones de los métodos son almacenadas fuera de la base de datos y esto puede ser demasiado complicado. Por ejemplo, si usted modifica la definición de una clase, todo el modelo, o aplicaciones que use en esta clase, deberán ser reenlazadas y redefinidas (redeplayed), sin embargo, al ejecutar las aplicaciones sin haber religado, mostrarán errores al no encontrar los métodos y habrá acceso a nuevas instancias, produciendo resultados fatales, y la posible corrupción de la base de datos. Este problema puede ser

resuelto a través del uso de versiones de objetos, lo que conlleva a una discusión más profunda, ya que al tener todas las versiones, asegura que los viejos métodos serán aplicados a las nuevas instancias, por lo que no habrá necesidad de reenlazar y redefinir las posibles aplicaciones afectadas.

Otro problema que existe al almacenar los métodos fuera de la base de datos, es que la seguridad y el control de concurrencia son sacrificados. En muchos sistemas contemporáneos, los métodos son almacenados en archivos estándar de UNIX, los cuales tienen ya un control de seguridad proporcionado por el sistema operativo, sin embargo, este enfoque no contempla el control de concurrencia sobre los métodos, lo cual incrementa la posibilidad de que los desarrolladores modifiquen al mismo tiempo algún método creando conflictos y con ello posibles inconsistencias en el acceso de los métodos a la base de datos. Por lo que este primer enfoque de almacenar los métodos en archivos externos no es el ideal, además de que rompe el principio de encapsulación y los métodos evitan la protección que brinda un DBMS.

La segunda alternativa almacena los métodos directamente en la base de datos, cumpliendo así con una de las premisas de la orientación a objetos," la encapsulacion", salvaguardando los métodos el DBMS.

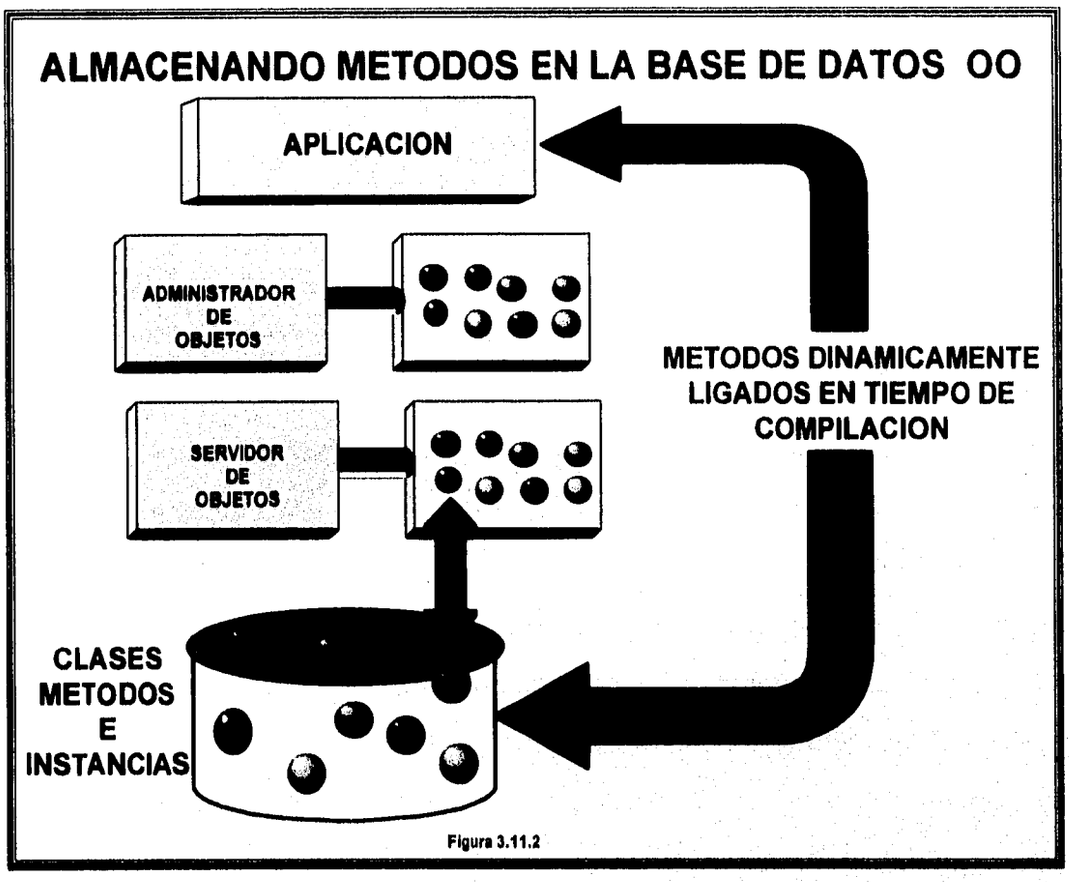


Figura 3.11.2

Este enfoque nos proporciona grandes ventajas :

- Los métodos pueden ser modificados automáticamente.- Si un método es modificado por un usuario, todas las aplicaciones que contienen ese método pueden accesarlo y utilizarlo inmediatamente. Cabe aclarar que no todos los sistemas lo pueden hacer, ya que para realizarlo se requiere de ligadura dinámica, la cual no está soportada por todos los lenguajes de programación OO, más sin embargo, casi todos los DBMS OO si lo contienen. Estos métodos son más seguros, ya que son almacenados directamente en la base de datos y dichos métodos se benefician de las bondades del DBMS en lo que se refiere a seguridad, por lo que los métodos almacenados en un DBMS comienzan a expresar inteligencia, ya que contienen sus propios procedimientos de operación o administración.

- Los métodos gozan del control de concurrencia, proporcionado por el DBMS, ya que como es bien sabido, una de las principales funciones de un DBMS es el control de concurrencia, logrando así mantener la integridad de la base de datos.
- Los métodos gozan de todos los beneficios de un DBMS, con este enfoque los métodos son tratados con el mismo valor que los datos, ya que no sòlo el DBMS proporciona seguridad de acceso y control de concurrencia, sino también reciben al igual que los datos, las rutinas de respaldo y administración de transacciones para múltiples cambios así como otros beneficios que generalmente proporciona un DBMS.

Este segundo enfoque como mencionamos en los tres puntos anteriores es mejor que el primero, sin embargo es de vital importancia el uso de el DBMS como contenedor de componentes reusables de software, en este sentido la principal responsabilidad del DBMS es el administrar la definición de las clases y desde luego los valores de las variables de instancia, por lo que es imprescindible que el DBMS prevea control para la administración de cambios, el control de concurrencia y los aspectos inherentes a la administración de la base de datos.

3.12 ADMINISTRACIÓN DE SEGURIDAD EN LAS TRANSACCIONES

En general las transacciones aíslan las acciones de un individuo relativas a la base de datos, hasta que se realiza la transacción con ésta, sólo entonces es visible globalmente. Las transacciones compartidas soportan grupos de usuarios en estaciones de trabajo, los cuales desean coordinar sus esfuerzos en tiempo real, de esta forma los usuarios pueden compartir sus resultados intermedios en una base de datos. La transacción compartida permite que varias personas intervengan en una sola transacción. Esto es útil en aplicaciones como Conferencias electrónicas y edición de documentos donde varias partes trabajan en el documento de manera concurrente. La seguridad puede ser a diferentes niveles dependiendo de cómo el DBMS las implementa, los objetos pueden ser asegurados en varios niveles de

- Los métodos gozan del control de concurrencia, proporcionado por el DBMS, ya que como es bien sabido, una de las principales funciones de un DBMS es el control de concurrencia, logrando así mantener la integridad de la base de datos.
- Los métodos gozan de todos los beneficios de un DBMS, con este enfoque los métodos son tratados con el mismo valor que los datos, ya que no sòlo el DBMS proporciona seguridad de acceso y control de concurrencia, sino también reciben al igual que los datos, las rutinas de respaldo y administración de transacciones para múltiples cambios así como otros beneficios que generalmente proporciona un DBMS.

Este segundo enfoque como mencionamos en los tres puntos anteriores es mejor que el primero, sin embargo es de vital importancia el uso de el DBMS como contenedor de componentes reusables de software, en este sentido la principal responsabilidad del DBMS es el administrar la definición de las clases y desde luego los valores de las variables de instancia, por lo que es imprescindible que el DBMS prevea control para la administración de cambios, el control de concurrencia y los aspectos inherentes a la administración de la base de datos.

3.12 ADMINISTRACIÓN DE SEGURIDAD EN LAS TRANSACCIONES

En general las transacciones aíslan las acciones de un individuo relativas a la base de datos, hasta que se realiza la transacción con ésta, sólo entonces es visible globalmente. Las transacciones compartidas soportan grupos de usuarios en estaciones de trabajo, los cuales desean coordinar sus esfuerzos en tiempo real, de esta forma los usuarios pueden compartir sus resultados intermedios en una base de datos. La transacción compartida permite que varias personas intervengan en una sola transacción. Esto es útil en aplicaciones como Conferencias electrónicas y edición de documentos donde varias partes trabajan en el documento de manera concurrente. La seguridad puede ser a diferentes niveles dependiendo de cómo el DBMS las implementa, los objetos pueden ser asegurados en varios niveles de

granularidad, virtualmente la mayoría de los productos soportan la seguridad de los objetos o un grupo de objetos con las llamadas páginas y segmentos. La definición precisa de estos términos varía, pero generalmente se refieren a cómo los objetos son organizados en disco. Las ventajas de la seguridad se brindan en cada nivel, la paginación y segmentación aseguran aún más la confiabilidad, en general se refiere a cómo los objetos son almacenados en el disco, los DBMS del mercado proveén de diferentes formas de mecanismos de la seguridad de los objetos, unos mediante la lectura y escritura de candados, lectura y escritura con notificación y otros mecanismos. Los DBMS OO debido a la complejidad de su información, requieren de nuevos y diferentes mecanismos de seguridad para el acceso a estructuras complejas y el óptimo funcionamiento de la base de datos.

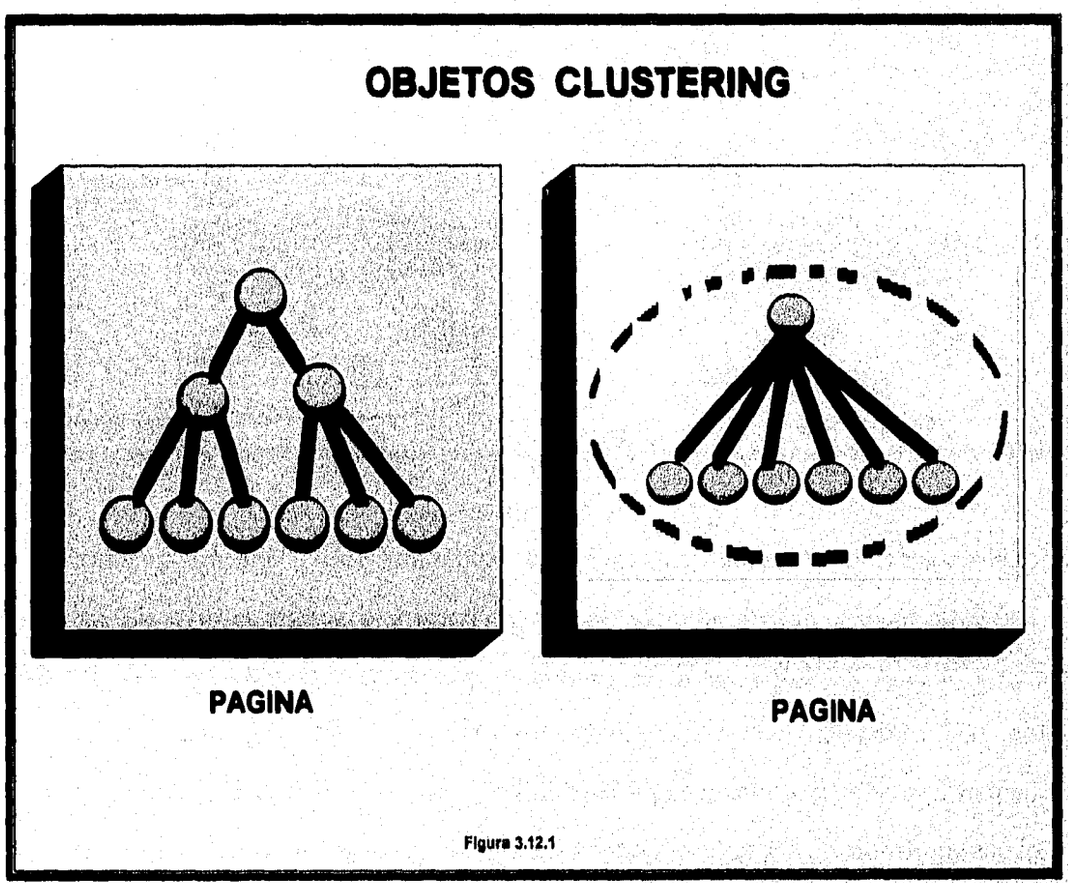


Figura 3.12.1

Manejo de Transacciones Cortas y Largas

Los DBMS normalmente proveén dos opciones de soporte para la seguridad, estas son conocidas como transacciones largas y transacciones cortas.

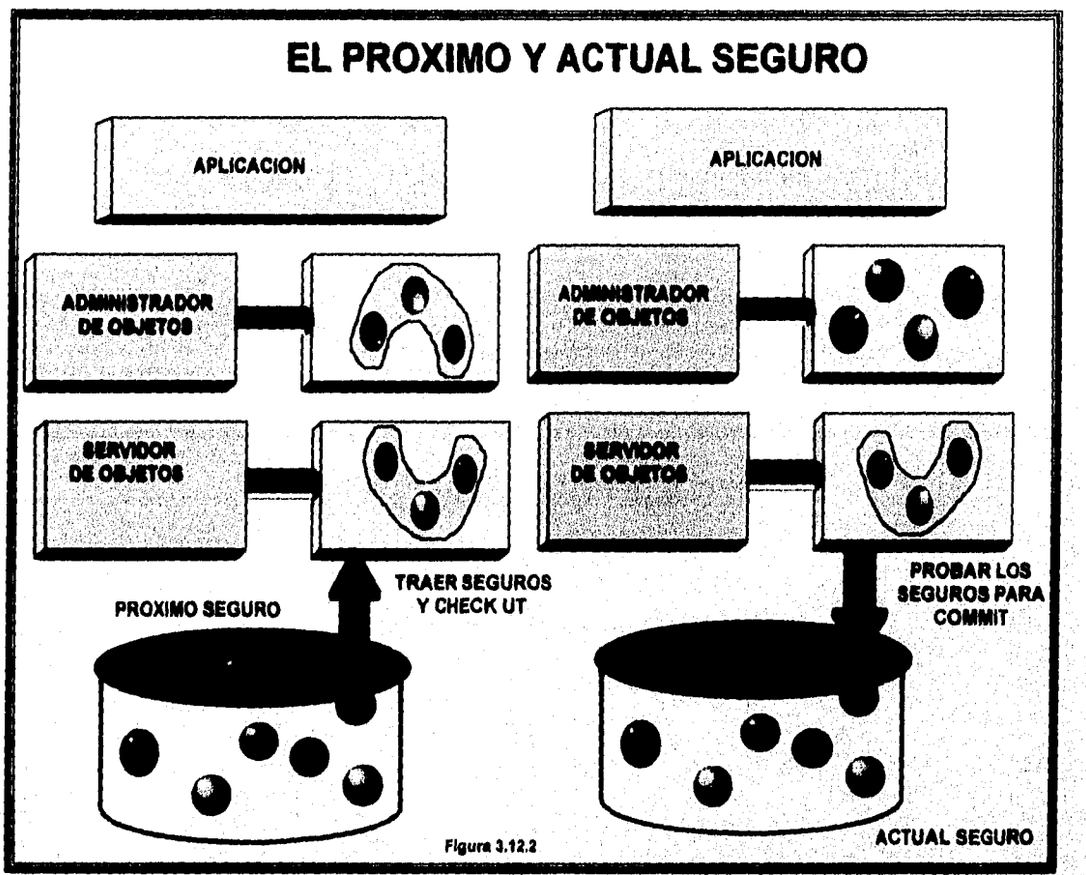


Figura 3.12.2

Los DBMS utilizan temporalmente rutinas de seguridad corta. Una transación corta es aquel proceso de seguridad en la cual la transación no sobrevive más allá de una sesión a la base de datos, es decir cuando el usuario se desconecta de la base de datos, la transación automáticamente es removida. Una transación corta es típicamente usada para realizar una modificación rápida a los valores almacenados, es el camino tradicional que la mayoría de los DBMS utilizan.

Las transaciones largas por el contrario, necesitan que las rutinas de seguridad sean persistentes, ya que deben sobrevivir a múltiples sesiones de base de datos, es decir

al terminar una sesión la rutina de seguridad sigue persistiendo y es ejecutada normalmente en la siguiente sesión. Las transaciones largas son utilizadas generalmente en aplicaciones del tipo CAD en las cuales un gran grupo de objetos son trabajados, es decir modificados, durante varias horas o incluso dias, para posteriormente regravarla en la base de datos, pero ya modificada.

DMSOO Distribuidos VS No Distribuidos

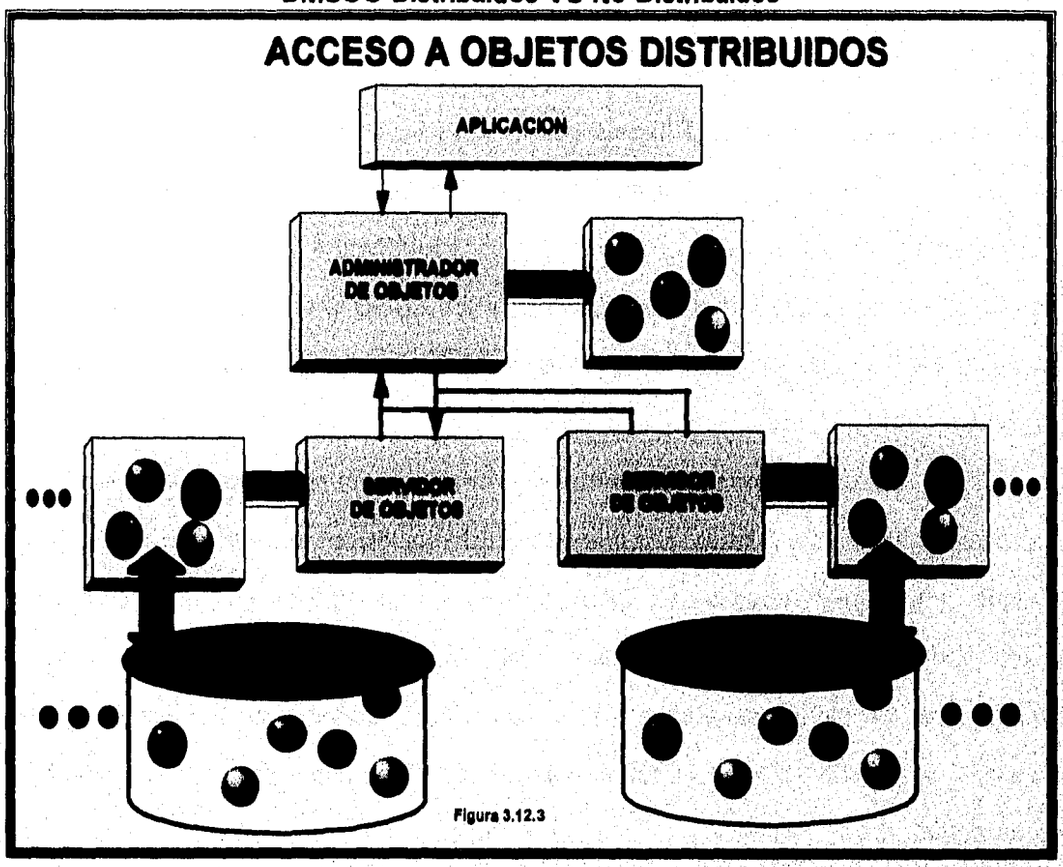


Figura 3.12.3

3.13 BASES DE DATOS ORIENTADAS A OBJETOS DISTRIBUIDAS

¿ Puede una base de objetos ser distribuida ?. Sí, en el sentido de que a menudo está formada por varios archivos, de suerte que estos archivos pueden residir en distintos puntos de la red, esto es sencillo.

¿ Puede ser de multi-usuario?. Esto depende del fabricante, requiere capacidad de escritura profunda, o cuando menos, poder poner candado contra la escritura a un objeto, una vez que ha sido ya solicitado por un proceso para escritura.

Además debe contar con un mecanismo de protección contra fallas, recuperación, a bordo de transacciones (rollback), etc. Algunos DBMSOO no cumplen con estos requerimientos, o lo solucionan muy trivialmente En el siguiente capítulo veremos las características generales de estos.

La independencia en la distribución es el último de una serie de pasos hacia la independencia que comenzó a finales de los años 60's con la independencia de los dispositivos y continuó a finales de los 70's principio de los 80's con la independencia de datos. Los temas más técnicos relativos a la realización de las BD Relacionales distribuidas se aplican también a las BDOO Distribuidas. Los sistemas de bases de datos distribuidas, se explican normalmente en un contexto del modelo cliente/servidor.

Se dice que una base de datos satisface el modelo cliente /servidor si la mayoría de los datos y la lógica relativa al procesamiento residen en uno o más servidores, aunque cada cliente procese sólo localmente los datos suficientes para ejecutar su query local en los servidores remotos y recibir los datos resultantes de la consulta provenientes de dichos servidores. El modelo cliente /servidor no implica una base de datos distribuida; un modelo cliente/servidor se distribuye solamente si tiene más de un servidor.

Existen grandes ventajas de la arquitectura cliente servidor. Una de ellas es que existe una mayor facilidad para imponer la integridad de los datos, el bloqueo y la seguridad, puesto que los datos se comprueban en la entrada y la salida de un único lugar. Otra ventaja es la mejor facilidad para ampliar los sistemas convirtiéndolos más grandes, añadiendo servidores y clientes. Además puede ser menos cara porque necesita menos potencia de procesamiento y capacidad de almacenamiento

para cada cliente. Finalmente existe un reducido tráfico de red, ya que el servidor devuelve solamente al cliente, el subconjunto de datos solicitado.

3.14 SOPORTE PARA WORKGROUPS

Casi siempre las BD personales son muy rápidas y convenientes por lo que algunos DBMSOO proveén la posibilidad de soportar bases de datos personales, adicionalmente por supuesto, al soporte para el grupo de bases de datos distribuidas, las ventajas de las bases de datos personales es que éstas se pueden accesar más rápidamente, ya que a través del overhead de la red son transferidos los mecanismos de control de concurrencia, rutinas de seguridad, y bypasses.

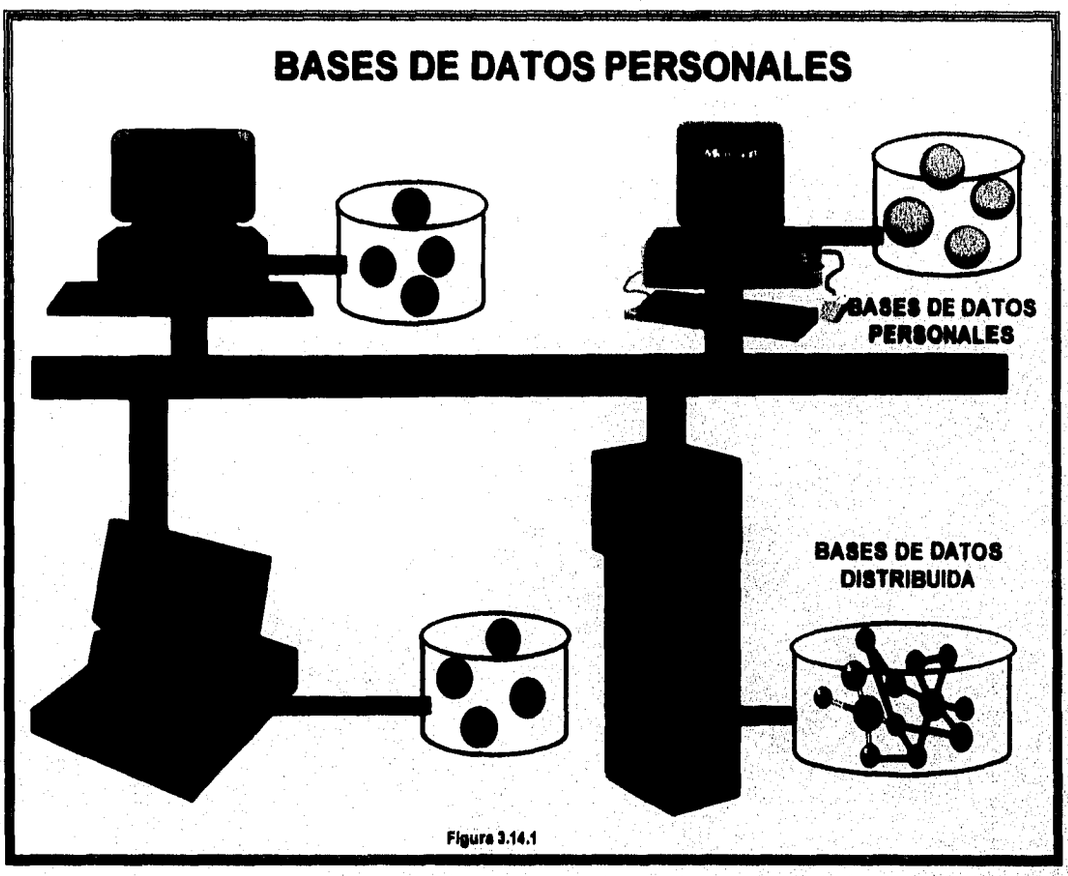


Figura 3.14.1

Las bases de datos personales son particularmente ventajosas en aplicaciones, donde los usuarios necesiten accesar grandes grupos de objetos por periodos

extendidos de tiempo. Estos requerimientos son típicamente soportados empleando procedimientos check-out, soportando transaciones largas, por ejemplo un programa CAD el cual checa fuera de los objetos una parte del diagrama, mismo que checa dos días antes de que fue revisado el diagrama.

3.15 VERSIONES DE OBJETOS

La mayoría de los DBMS sólo permiten que exista una representación de un ente de la base de datos dentro de ésta, las versiones permiten que las representaciones alternas existan en forma simultánea, lo cual es útil en diversas áreas de aplicación en donde existan diseños en evolución y se manejen estrategias diferentes en forma recurrente (ejemplo: presupuestos, diseño VLSI, etc.). Los diseñadores de un sistema podrían tener varias versiones de un subsistema particular para probar diversos algoritmos y poder evaluar el desempeño general del sistema con cada uno de ellos.

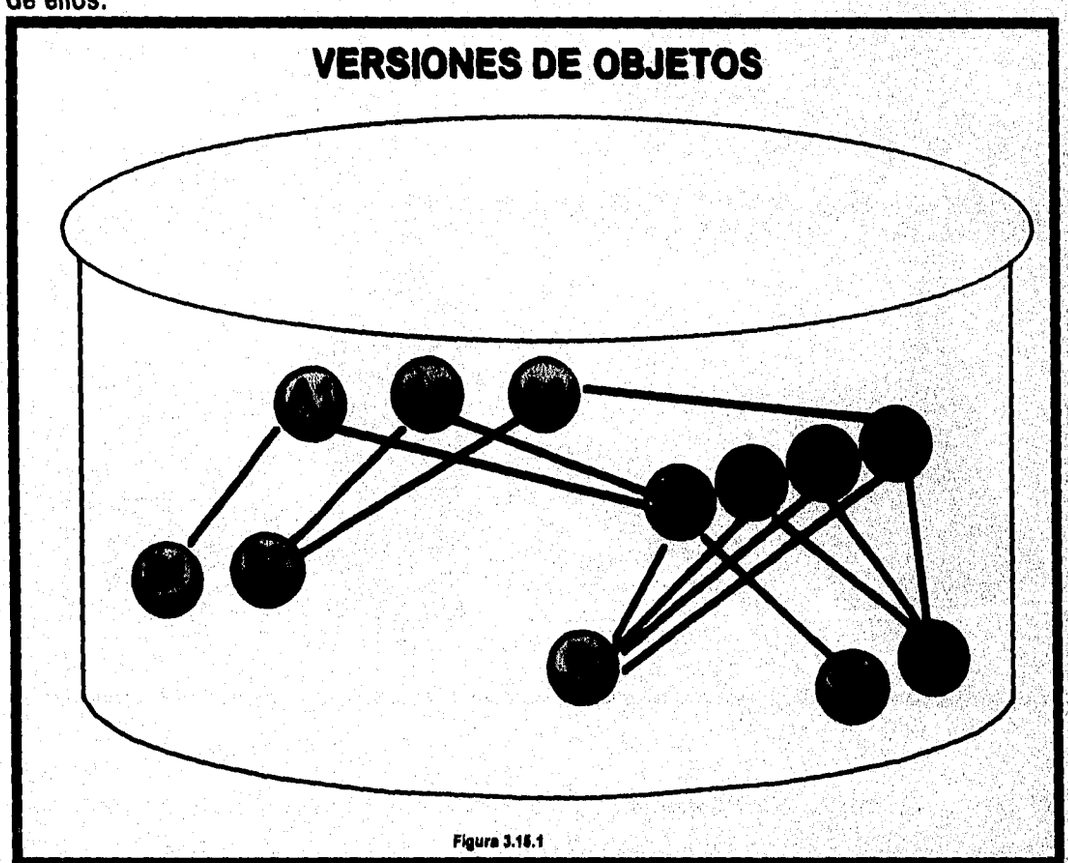


Figura 3.15.1

El control de versiones permite diferentes alternativas. Otro importante uso de las versiones, es proveer de alternativas de solución a los mecanismos de diseño, por ejemplo se pueden almacenar las alternativas con conflictos y una vez superados éstos, el diseño se hará en base a la decisión adecuada, el usuario determina que versión borra y cuales seguirán persistentes.

3.16 DESARROLLO CON BDOO

Las BDOO se desarrollan al describir en primer lugar los tipos de objetos importantes del dominio de la aplicación y los comportamientos asociados con aquellos tipos de objetos. Estos objetos determinan las clases que conformarán la definición de la BDOO, ejemplo una base de datos diseñada para almacenar la geometría de ciertas partes mecánicas, incluiría clases como cylinder (cilindro), sphere (esfera) y cube(cubo). El comportamiento del cylinder podría incluir información relativa a sus dimensiones, volumen y área superficial:

```
CLASS CYLINDER {
        FLOAT HEIGT();          //* altura
        FLOAT RADIUS();         //* radio
        FLOAT VOLUME();         //* volumen
        FLOAT SURFACEAREA(); //* área superficial
        };
```

Se puede llegar a definiciones similares para el cubo y la esfera. En la definición anterior **HEIGHT()** (altura), **RADIUS()** (radio), **VOLUME()** (volumen), **SURFACEAREA()**; (área superficial), representan los mensajes que se pueden enviar a un objeto **cylinder**. Como se puede observar, no existen detalles de implantación, aún si se almacena o calcula la información. La implantación se efectúa en el mismo lenguaje, escribiendo funciones correspondientes a las solicitudes OO:

```
CYLINDER :: HEIGT() RETURN{ CYLINDER_HEIGHT;}
CYLINDER :: VOLUME() RETURN{ PI*RADIUS()*RADIUS()* HEIGHT();}
```

En este caso **HEIGHT** se almacena como un elemento de datos, mientras que **volume** se calcula mediante la fórmula apropiada. Observe que la implantación interna de **volume** utiliza solicitudes para obtener **RADIUS()** y **HEIGHT()**; sin embargo, el aspecto más importante es la sencillez y uniformidad que experimentan los usuarios de **CYLINDER**. Sólo necesitan conocer la forma de enviar una solicitud y las solicitudes disponibles. Toda la aplicación se puede escribir con un estilo uniforme, por lo que ya no se necesita la separación del lenguaje de programación y el DML. Este enfoque también le dá flexibilidad a la BDOO, puesto que toda la aplicación se escribe por medio del envío de solicitudes, las implantaciones se pueden alterar en cualquier grado, sin afectar a las aplicaciones. En consecuencia, la aplicación es más sencilla y más fácil de mantener y mejorar. En este sentido las BDOO ofrecen una separación aún mayor entre la especificación de un sistema y su implantación.

Los Enfoques Para la Construcción de una BDOO

Las BDOO se pueden construir mediante tres enfoques.

Uno de esos enfoques utiliza un sistema de administración convencional de una base de datos y añade un nivel para el procesamiento de las solicitudes y los métodos de almacenamiento OO, el mérito de este enfoque es que se puede reutilizar el código actual, altamente complejo para los DBMS tradicionales, de modo que una BDOO se implanta a partir de otras existentes sin partir totalmente de cero.

El segundo enfoque considera a la BDOO como una extensión de la tecnología de las bases de datos relacionales, de esta forma las herramientas técnicas y hasta la experiencia de la tecnología relacional se utiliza para la construcción de DBMS Relacionales extendidos. Se puede añadir apuntadores a las tablas relacionales para ligarlas con los objetos binarios de gran tamaño BLOB. Varios proveedores de software apoyan este enfoque.

El tercer enfoque reflexiona acerca de la revolución de la tecnología y construye los DBMSOO puros, los proveedores de esta tecnología aseguran que la tecnología relacional es un subconjunto de una capacidad más general que la OO, y las estructuras relacionales se pueden utilizar en los casos adecuados pero para el manejo de estructuras complejas es mejor cambiar el modelo relacional por el OO, ya que las BDOO permiten el acceso a los datos de un objeto complejo sin desplazar el mecanismo de acceso. Los proveedores de DBMSOO Puros afirman que son dos veces más rápidos que los DBMS Relacionales para almacenar y recuperarar información compleja, por lo tanto son esenciales en aplicaciones como CAD, y permitir que un depósito CASE fuera una facilidad de tiempo real en vez de una facilidad por lotes.

3.17 PROBLEMÁTICA EN LAS BDOO

El problema principal que tienen que resolver las bases de datos orientadas a objetos es el de convertir las referencias a otros objetos, que en memoria principal son direcciones de memoria,de forma tal que sean "permanentes", pues cuando estos objetos vuelvan a ser cargados en alguna otra ocasión a otro espacio de direccionamiento de memoria principal o en algún otro proceso, desde luego no ocuparán la misma posición de memoria. Dicho de otra manera, el problema estriba en cómo guardar los valores almacenados en cada ranura, y cómo guardar los métodos. En memoria principal las ranuras contienen dos tipos de datos:

a) Los datos primitivos (cadenas, enteros, números reales, etc.) se guardan ellos mismos en la ranura, es decir no se guarda un apuntador a ellos (una posible excepción son las cadenas).

b) En cambio cuando una ranura guarda (tiene como valor) a un objeto o a un conjunto de ellos, este valor se guarda como un apuntador al lugar de la memoria donde está almacenado el objeto.

Es decir, cada objeto (pero no cada dato primitivo) se guarda en un lugar único en memoria, y se puede hacer referencia a tal objeto por medio de un apuntador al

mismo. Desde este punto de vista, los objetos no son más que un identificador (el nombre del objeto) más un manojo de ranuras, las que contienen ya sea datos primitivos o apuntadores a otros objetos. Como tal, estas estructuras parecería que fuera fácil copiarlas tal cual a disco, y traerlas después cuando se les necesite nuevamente.

El problema está en que esos objetos, al ser leídos nuevamente (en alguna otra corrida del programa, o en otro programa) del disco a la memoria, no van a ser cargados necesariamente en la misma dirección del espacio de direccionamiento donde antigüamente estaban. Por consiguiente, los apuntadores van a volverse inválidos, pues no van a apuntar a la nueva dirección de carga de cada objeto, sino a la dirección antigüa.

La solución en general consiste en no usar apuntadores en lo que se guarda en el disco.

Hay cuando menos dos soluciones posibles:

a) Convertir los apuntadores (que son direcciones de memoria principal) a cada objeto, por el nombre en ascii del objeto apuntado, si previamente se exigió que cada objeto tenga un nombre único. La cadena en ascii (el nombre del objeto) se precede por algún símbolo especial que denote "yo no soy una cadena cualquiera, yo realmente substituyo a un apuntador, a un objeto cuyo nombre ahora poseo". Cada ranura de cada objeto, al guardarse en disco convierte su valor como sigue: los apuntadores a objetos como ya se describió; los números, cadenas y otros objetos primitivos, a notación ascii convencional. Al leerse la memoria, los objetos primitivos pasarán a ocupar directamente su lugar en la ranura, en tanto que los "apuntadores ascii a objetos" serán convertidos a una dirección de memoria donde el referido objeto (cuyo nombre lleva) esté guardado, guardándolo si es necesario en caso de que no se encuentre. Por este motivo, todos los objetos que vienen de (o van a) disco pasan por una tabla de dispersión (tabla hash) tabla de objetos, que convierte apuntadores a objetos (dirección en memoria principal donde se encuentra el objeto depositado) a cadenas ascii (nombre del objeto).

Convirtiendo la dirección de cada objeto (un apuntador a memoria principal) a un texto en ascii, es decir, el nombre del objeto referido. Hay que tener cuidado de no convertir así a los números y cadenas que el objeto contiene (la edad del objeto Pedro Rivera, digamos, es un número), sino grabarlas tal como son a la base permanente. En todo caso, cadenas, números y demás objetos simples tienen un descriptor para identificarlos plenamente como tales. En muchos casos esta información está asociada a la propiedad (edad, en el ejemplo cuyo valor se guarda).

b) La otra solución consiste en usar un espacio de direccionamiento independiente del de la memoria principal (es decir, las direcciones en este espacio no son idénticas con las de la memoria principal, pero hay una correspondencia uno a uno entre ellas). Cada apuntador a memoria principal, llamado apuntador local, se convierte a un apuntador "independiente", llamado apuntador global o virtual antes de ser almacenado en disco. De nuevo se usa una tabla de objetos para hacer esta conversión, y guardándolo en disco junto con una bandera que diga "yo soy un apuntador virtual".

La razón de llamarse "apuntador local" es que se refiere sólo a los objetos que están en memoria principal. En aplicaciones que tienen muchos objetos, todos ellos se encuentran en la base de objetos (en disco), en tanto que sólo una parte de ellos se trae a memoria. Por este motivo también, se les llama apuntadores globales a aquellos que se encuentran almacenados en disco.

La conversión de apuntadores locales a globales (o apuntadores ascii) la realizan de manera automática los DBMSOO.

INTERACCIÓN ENTRE LOS OBJETOS PERMANENTES Y AQUELLOS EN MEMORIA

Básicamente hay que tomar nota de cuales objetos han sido traídos a memoria, y cuales aún están en disco. Con este fin, la tabla de objetos que existe en memoria se amplía para indicar que algunos objetos estaban en memoria pero ahora están en disco, en tanto que otros son referidos en memoria (por el apuntador virtual

digamos) pero no está en memoria. Ontos ODBMS utiliza objetos suplentes (surrogate objects) en memoria, que apuntan a sus titulares, si ellos están en memoria también, o null si no están.

Si no se han traído todos los objetos a memoria, es posible que algunos de los ya traídos apunten a otros que aún están en disco. Es decir, es posible que contengan apuntadores globales. Debe haber, pues, una manera de identificar si un apuntador es local o global, si se desea permitir que sólo parte de los objetos se bajen a memoria de disco. Una manera es usar apuntadores grandes para los apuntadores globales, y menos bits para los locales. Otra manera utilizada por Ontos es que los objetos globales apuntados desde la memoria, se substituyen (en memoria) por suplentes, que son objetos "flacos" porque sus ranuras acrecen de valor (ya que éstos no se han leído de memoria). Cuando se desea accesar un objeto y en vez de él se accesa al suplente, éste se encarga de traer a su titular (es decir la información de sus ranuras) de disco. El objeto suplente apuntará entonces a su titular, apuntaba a NIL cuando éste no estaba en memoria.

Esta interacción entre objetos en memoria y en disco la realiza automáticamente el DBMSOO.

Los **Objetos en memoria**. Ejemplo: C++

- Cada objeto tiene un lugar único en memoria. (Esto permite manejar apuntadores a los objetos simplemente).
- Dichos apuntadores son a direcciones de memoria, válidas solamente dentro de "éste" espacio de direccionamiento.
- Las diferencia es que una base de datos no guarda apuntadores.

La **tabla de objetos** conoce la posición en memoria de cada objeto y se usa después para saber si un objeto existe pero no está en memoria, además la tabla de objetos asocia los nombres del objeto (una cadena de caracteres) con el objeto mismo (un apuntador a donde está guardado en memoria).

digamos) pero no está en memoria. Ontos ODBMS utiliza objetos suplentes (surrogate objects) en memoria, que apuntan a sus titulares, si ellos están en memoria también, o null si no están.

Si no se han traído todos los objetos a memoria, es posible que algunos de los ya traídos apunten a otros que aún están en disco. Es decir, es posible que contengan apuntadores globales. Debe haber, pues, una manera de identificar si un apuntador es local o global, si se desea permitir que sólo parte de los objetos se bajen a memoria de disco. Una manera es usar apuntadores grandes para los apuntadores globales, y menos bits para los locales. Otra manera utilizada por Ontos es que los objetos globales apuntados desde la memoria, se substituyen (en memoria) por suplentes, que son objetos "flacos" porque sus ranuras acrecen de valor (ya que éstos no se han leído de memoria). Cuando se desea accesar un objeto y en vez de él se accesa al suplente, éste se encarga de traer a su titular (es decir la información de sus ranuras) de disco. El objeto suplente apuntará entonces a su titular, apuntaba a NIL cuando éste no estaba en memoria.

Esta interacción entre objetos en memoria y en disco la realiza automáticamente el DBMSOO.

Los **Objetos en memoria**. Ejemplo: C++

- Cada objeto tiene un lugar único en memoria. (Esto permite manejar apuntadores a los objetos simplemente).
- Dichos apuntadores son a direcciones de memoria, válidas solamente dentro de "éste" espacio de direccionamiento.
- Las diferencia es que una base de datos no guarda apuntadores.

La **tabla de objetos** conoce la posición en memoria de cada objeto y se usa después para saber si un objeto existe pero no está en memoria, además la tabla de objetos asocia los nombres del objeto (una cadena de caracteres) con el objeto mismo (un apuntador a donde está guardado en memoria).

Los **Objetos en disco**. Ejemplo: Ontos.

Convirtiendo direcciones de memoria en direcciones globales o virtuales.

También se podrían haber convertido los apuntadores a memoria a cadenas (remplazando en disco un objeto por su nombre en ascii).

Se lleva nota de:

Los objetos que han sido traídos a memoria.

Los que existen pero están en disco.

Los que no existen.

Dos regímenes de migración de objetos de la base de datos a memoria (recordar, es memoria virtual en todo caso).

Traer todos los objetos desde el principio.

Traer todos los objetos conforme se vayan necesitando (refiriéndose a ellos en el programa de aplicación).

Bases de Datos OO Distribuidas.

El problema de compartir objetos entre dos procesos: Un proceso se refiere al objeto "Pedro Rivera" mediante una dirección de memoria, que no es la misma que otro proceso usa; sin embargo, ambos están hablando del mismo objeto. Con las bases de datos Orientadas a Objetos es posible que varios procesos (cada uno con su espacio de direccionamiento) compartan objetos de una misma base de objetos. En primer lugar si los procesos no se comunican entre ellos y no comparten objetos comunes, esto implica que no hay interacción alguna entre tales procesos, por lo que cada uno procede independientemente con respecto a la actualización y uso de la base de objetos.

Ahora bien, supongamos que los procesos se comunican entre ellos o comparten ciertos objetos. Si un proceso le va a mandar un objeto a otro (o se va a referir a algún objeto, como argumento de una función para que el otro proceso la ejecute), el remitente debe convertir su apuntador (local) a un apuntador "global" (o a un nombre ascii) para que el objeto destinatario le entienda y pueda descifrar de qué objeto se trata, ya que los apuntadores locales no son entendibles (no tienen

sentido) para otro proceso que no sea el local. Sólo a través de los apuntadores globales (o de los apuntadores en ascii) se pueden comunicar los procesos con estos objetos.

Cuando se está compartiendo el mismo objeto por varios procesos, existe el riesgo de que los cambios de uno no los vea el otro. Las bases de objetos por lo general se van por el lado fácil. Cuando un proceso extrae ciertos objetos de la base de objetos (en disco), ningún otro proceso los puede modificar, están "prestados" al primero que los "sacó" de disco a memoria. Como sólo un proceso tiene la versión actual del objeto (los otros tienen copias viejas sólo de lectura), todos los procesos excepto uno están trabajando con un "objeto viejo" cuyos valores pueden haber cambiado ya substancialmente.

Para evitar lo anterior, se usa la técnica de la escritura profunda (write-through); cuando un objeto cambia de valor, éste se transmite a la base de objetos (en disco) y de ahí a todos los demás poseedores de tal objeto. Así esta implementado C y C++.

3.18 LENGUAJES DE CONSULTA AD-HOC

La principal operación solicitada por los usuarios es la consulta, en la información almacenada en la base de datos. Una diferencia fundamental entre las BDR y las BDOO es la cantidad de información que se mueve entre la aplicación y el DBMS, las aplicaciones envían consultas a la base de datos relacional, la cual devuelve entonces un número de valores, estos valores son almacenados normalmente como parte de las estructuras de datos de la aplicación, manipulados y vueltos a entregar de nuevo a la BDR para su almacenamiento. En contraste con ello, cuando las aplicaciones envían mensajes a las BDOO, el DBMSOO manipula los datos con los métodos, recupera o calcula un valor, y devuelve el valor a la aplicación.

Existe mucho más tráfico entre una aplicación y una BDR que almacena objetos. En una BD Relacional de este tipo, los objetos complejos son divididos y almacenados como campos en tablas independientes, de modo que al solicitar la aplicación el

objeto, un número de consultas debe aplicarse para recuperar y volver a ensamblar un sólo objeto, estas consultas se aplican secuencialmente de modo que una consulta depende del resultado de la anterior. La base de datos Relacional no conoce la petición global; conoce solamente las consultas individuales y por consiguiente no puede reordenar las consultas para optimizarlas. En una BDOO Pura, un sólo mensaje toma el lugar de muchas consultas de una base de datos relacional. Un mensaje puede solicitar cálculos y hacer que se envíen mensajes a otros objetos antes de entregar un resultado. Esto alivia el problema de la necesidad de consultas secuenciales.

El SQL normado por ANSI se ha convertido en un estándar para realizar consultas en bases de datos relacionales. Por lo que las BDOO han tenido que incorporar el SQLOO para poder hacer más fácil la migración de BDR a BDOO. A pesar de que la estructura relacional y el SQL no son consistentes con las estructuras de una base de datos orientada a objetos. Las consultas según el predicado relacional se dirigen contra estructuras desconocidas, pero la consulta orientada a objetos navega a través de punteros que definen una estructura conocida. Para resolver estas diferencias, las bases de datos orientadas a objetos utilizan dos métodos principalmente:

- Se amplían los tipos conocidos por la sintaxis SQL para permitir que sean consultados los objetos y también las tablas;
- Se emiten las consultas SQL y también las de objeto en tándem y posteriormente unir las dos consultas en la aplicación.

El problema principal con las extensiones de SQLOO ha sido el rendimiento; el proceso de coincidencia y búsqueda relacional es intrínsecamente lento cuando se le compara con una base de datos puramente orientada a objetos en la que las consultas referencian directamente a los objetos.

Al trabajar con grandes cantidades de datos es imperativo poder buscar en la base de datos tan eficientemente como sea posible. El rendimiento de una base de datos relacional depende en gran medida de la optimización automática de la consulta.

Dicha optimización es necesaria para que las BDOO sean eficientes, en las BDR, la optimización de la consulta explota las propiedades del álgebra relacional y la estructura física de las relaciones (tablas). Por ejemplo es conocido que se conmutan la selección y la unión. La heurística[V] puede aplicarse debido a la estructura uniforme a lo largo de todos los datos. Una premisa heurística, por ejemplo, podría ser "Seleccionar en primer lugar por los campos para los cuales existen índices". Esto reducirá el conjunto en el que debe realizarse una unión.

En un sistema Orientado a objetos, no puede haber un conjunto estándar de operaciones para todos los datos, ya que las operaciones o sea los métodos se definen por tipo. Además no hay forma de determinar si se van a conmutar dos operaciones cualquiera por ello no son útiles las técnicas estándar de optimización de consultas. Una técnica propuesta denominada **revelación**, podría con estas dificultades

La técnica de la revelación permite al optimizador comprobar en el interior de las definiciones de clase e indagar, de otra forma detalles ocultos de realización. Aunque esta estrategia viola el principio del Paradigma de La OO (de la privacidad en la realización de las clases encapsulación), es difícil imaginar cómo el optimizador de consultas podría lograr sus objetivos de otra forma.

Los DBMSOO son construidos con la idea de utilizar el mecanismo de navegación a través de punteros para poder accesar los datos,.en contraste con los DBMSR que están basados en el concepto de acceso a los datos a través de la asociatividad de las tablas. Los DBMSOO actualmente representan el regreso a los viejos modelos jerárquicos y de red los cuales para su acceso están basados en la navegación.

Sin embargo, los vendedores de DBMSOO reconocen la importancia del acceso asociativo y virtualmente todas las facilidades que brinda este tipo de acceso. A la fecha los accesos asociativos en los DBMSOO no son muy rápidos ni eficientes comparados con los DBMSR, pero mientras madura esta tecnología con ese tipo de

[V] Ver glosario de terminos.

limitaciones en su arquitectura. Los DBMSOO finalmente serán ajustados en algunos procesos al funcionamiento de acceso asociativo como en los sistemas relacionales.

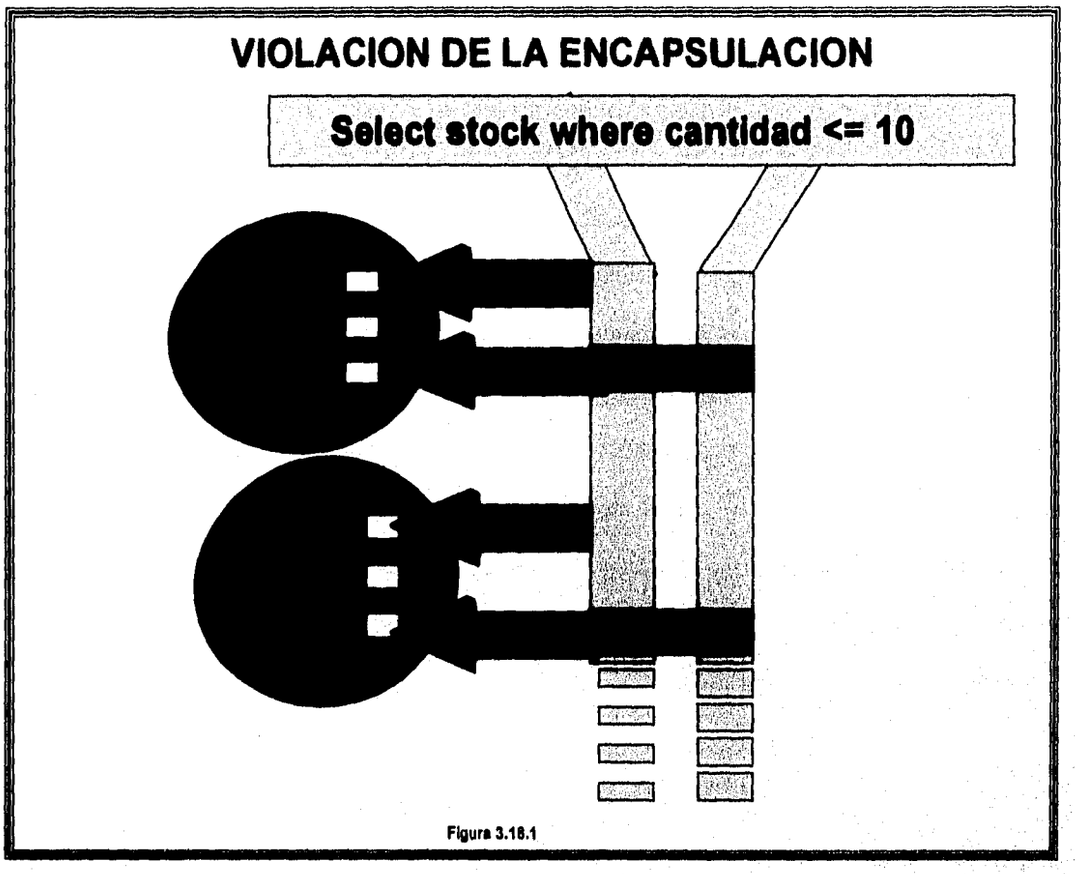


Figura 3.18.1

Por supuesto una de las ventajas del acceso asociativo es la habilidad de los lenguajes de consulta especialmente el estándar SQL. Mientras que SQL no es naturalmente adecuado para la recuperación de objetos, más vendedores de DBMSOO van extendiendo el modelo de SQL, creando un SQLOO para accesar estos sistemas. El último de los vendedores tiene SQLOO en el mercado actual, y un esfuerzo informal es ahora obsoleto para estandarizar esta extensión del SQL. Aunque algunos vendedores de DBMSOO incorporan SQLOO para agregar acceso a los sistemas a través de asociatividad. Esto viola uno de los principios básicos de la tecnología de orientación a objetos "la encapsulación de los datos". En orden para traer razonablemente el contenido basado en acceso asociativo, el método es

totalmente de by-pass examinando las variables directamente construyendo B-trees y tablas hash, y otras estructuras basadas en la busca de los valores. Este compromiso de valuaciones de encapsulación por los cambios en las estructuras de datos, no son muy largos aisladores del objeto, pero requiere cambios en el código de acceso asociativo que ha tenido modificación de acceso en estas estructuras.

Es evidente la necesidad de continuar ofreciendo un lenguaje de consulta, tomando en consideración el modelo OO, que permita extraer objetos o parte de ellos a través de un lenguaje sencillo, de tipo declarativo como SQL. Por otra parte en el modelo OO habrá casos en que existan enlaces entre dos tipo de objetos, los cuales pueden recuperarse directamente, mientras que en el modelo relacional habría sido necesaria una unión. En las BDOO son posibles estructuras grandes y complejas, y podrían convertir en innecesarias algunas optimizaciones. Por ejemplo, las consultas tales como seleccionar todas las piezas que constituyen el componente XYZ se gestionan fácilmente en una BDOO ya que cada objeto componente se almacena junto con toda la información de sus piezas.

3.19 INTEGRIDAD DE LOS DATOS

La integridad se entiende como la propiedad que asegura la calidad de los datos, respetando las propiedades o reglas especificadas en su definición. La integridad global de las bases de datos incluye aspectos como: integridad semántica, (fidelidad en la descripción de los objetos del mundo real), seguridad en el acceso (confidencialidad y protección de los derechos de los usuarios) y seguridad en el funcionamiento (recuperación en caso de caída). Todos estos aspectos que constituyen la integridad de la base de datos, han requerido una adaptación al modelo OO y se han planteado problemas tales como: especificación de reglas sobre objetos, establecimiento de candados sobre objetos complejos o porciones de ellos, definición de protecciones y autorizaciones de accesos sobre los objetos, muchos de estos problemas han sido actualmente abatidos por los DBMSOO, y otros están en proceso de desarrollo.

Cada programa que accede a una base de datos es una amenaza potencial para la integridad de la base de datos. Los DBMS están a la defensiva contra estas amenazas proporcionando restricciones para salvaguardar la integridad de los datos, o condiciones que deben obedecer siempre los elementos de datos. El ejemplo de una restricción de la integridad es una restricción sobre los valores de los elementos de los datos, es decir, exigir que la edad de un empleado oscilé entre 0 y 100. Otro ejemplo es una restricción en el número de elementos de datos, exigiendo que si el registro de un estudiante muestra una inscripción para el curso de Robótica 777, dicho curso debe figurar en la base de datos del curriculum, estas restricciones de integridad se aplican por igual a las BD Relacionales y a las BDOO. Sin embargo, existe un tipo adicional de restricción de integridad en las BDOO denominado restricción a nivel de variable modelo, que se deriva del paradigma OO. Una defensa a nivel de variable modelo puede aplicarse a un subconjunto o subclase de datos, a diferencia de la mayor parte de las defensas de integridad que se aplican a todos los elementos de datos del mismo tipo. Supongamos que un empleado particularmente valioso va a ser recompensado con un salario que excede el límite que debe cumplir el salario de los demás empleados. Es conveniente agregar una defensa a nivel de variable modelo para el salario de este empleado, por medio del mecanismo de la subclasificación sin alterar la limitación de salario para los demás empleados. Los DBMS Tradicionales no soportan esta consideración.

Un medio utilizado por las BDOO para bloquear violaciones a las restricciones, es un mecanismo de excepción, el cual cuando encuentra una situación excepcional en la BDOO, crea una excepción produciendo un objeto de excepción y transfiriendo el control al administrador de excepciones correspondiente. El objeto excepción puede entonces ser interrogado por el administrador de excepciones para conocer la situación excepcional. Como una excepción es un objeto, éste tiene un tipo y permite por tanto el agrupamiento por tipos de condiciones similares de excepciones, como por ejemplo: Divide by zero (División entre cero) y Overflow

(desbordamiento), subtipos ambos de Arithmetic Error (Error aritmético), con un administrador de excepciones adecuado para cada tipo.

Otro medio de proporcionar restricciones a la integridad es por medio de disparadores, los cuales son mecanismos conectados o agregados a determinados elementos de datos dentro de una base de datos que se activan, siempre que se produce un intento de acceso o modificación de los elementos de datos. Los disparadores comprueban que no se esta haciendo nada ilegal o incorrecto en los elementos de datos, los disparadores realizan también actualizaciones para hacer consistente a la base de datos. Consideremos una base de datos con registros de jefes y empleados. Siempre que se actualiza al jefe de un empleado, un disparador puede ajustar automáticamente la lista de informes directos para el antiguo y el nuevo jefe. Las BDOO proporcionan un método único para realizar los disparadores efectuando el seguimiento de las llamadas o invocaciones a los métodos. Como los datos pueden accederse únicamente a través de los métodos, las bases de datos OO pueden identificar los intentos de modificación de los datos efectuando el seguimiento de aquellos métodos que tienen acceso a los datos e invocando un disparador adecuado siempre que se invoque un método. En BD muy grandes resulta con frecuencia ineficaz la comprobación de todos los disparadores y demás restricciones de la integridad antes de realizar una actualización. En concreto, en una BDOO en la que cualquier operación puede ser definida por el usuario, el sistema no puede determinar fácilmente por adelantado cómo podrán ser afectadas otras operaciones. El problema está en localizar aquellos punteros críticos en los que deben activarse las defensas de la integridad. Es un problema general que está pendiente todavía por resolverse.

CONCURRENCIA

Las realizaciones tradicionales para garantizar la integridad de la base de datos durante las operaciones concurrentes, se basan generalmente en la regla de que cada elemento de datos que se va a leer o escribir durante una transacción debe estar disponible para el resto de transacciones. Por ejemplo, antes de poder actualizar un registro de un empleado, la BD debe obtener acceso exclusivo a dicho registro. En las transacciones sencillas este tipo de gestión de transacciones a nivel de lectura/escritura no es un problema, sin embargo, el acceso a un registro de una aplicación CAD puede suponer el acceso a un gran número de registros de datos interrelacionados. Si la transacción debe obtener acceso exclusivo a todos estos registros, puede degradar el funcionamiento de otras transacciones concurrentes del sistema.

LOS ESQUEMAS DE RECUPERACIÓN

Una de las principales bondades de la orientación a objetos aplicada a las bases de datos es su enfoque para el modelado del esquema conceptual, reduciendo la distancia semántica entre los objetos reales, los objetos representados en la computadora y los objetos almacenados. El modelo OO representa de manera más cercana y fiel el dominio del problema, mapeando las abstracciones de las entidades del mundo real en objetos de la base de datos facilitando así, la conceptualización y solución de los problemas planteados. Además resulta de gran importancia para el desarrollador contar con un ambiente que permita la representación de entidades del mundo real como objetos de la base de datos, y facilite su manipulación sin necesidad de desmenuzarlos para ser almacenados y reconstruidos cuando se requiera consultarlos. Cada entidad del mundo real es modelado por un tipo de objeto. A cada objeto se le asocia un identificador único, el que será almacenado como una tupla de los identificadores de sus partes.

3.20 BASES DE DATOS RELACIONALES VS BDOO

Aquí está un ejemplo de cómo es más eficiente un DBMSOOs y puede ser menos eficiente un DBMS-Relacional al almacenar información compleja. La naval de U.S.A. esta desarrollando un sistema para adquisición de materiales que requieren un mantenimiento distinto, originalmente para cumplir este propósito se buscó un DBMS basado en partes de información almacenadas en una base de datos, pero todos son muy lentos para manipular estas estructuras de datos. Por ejemplo, almacenar la geometría necesaria para desplegar e imprimir una tarjeta de circuitos integrados consistente de 2500 componentes en un sistema CAD requiere de un cuarto de hora con un DBMS relacional. Para ver si los DBMSOOs pueden ayudar con este problema, la naval cargó la misma información en un DBMSOO e hicieron una prueba de comparación. El DBMSOO realizó el mismo proceso en sólo nueve segundos, una rápida mejora de magnitud dos. El mismo grado de mejora fue obtenido con reintegración como tal.

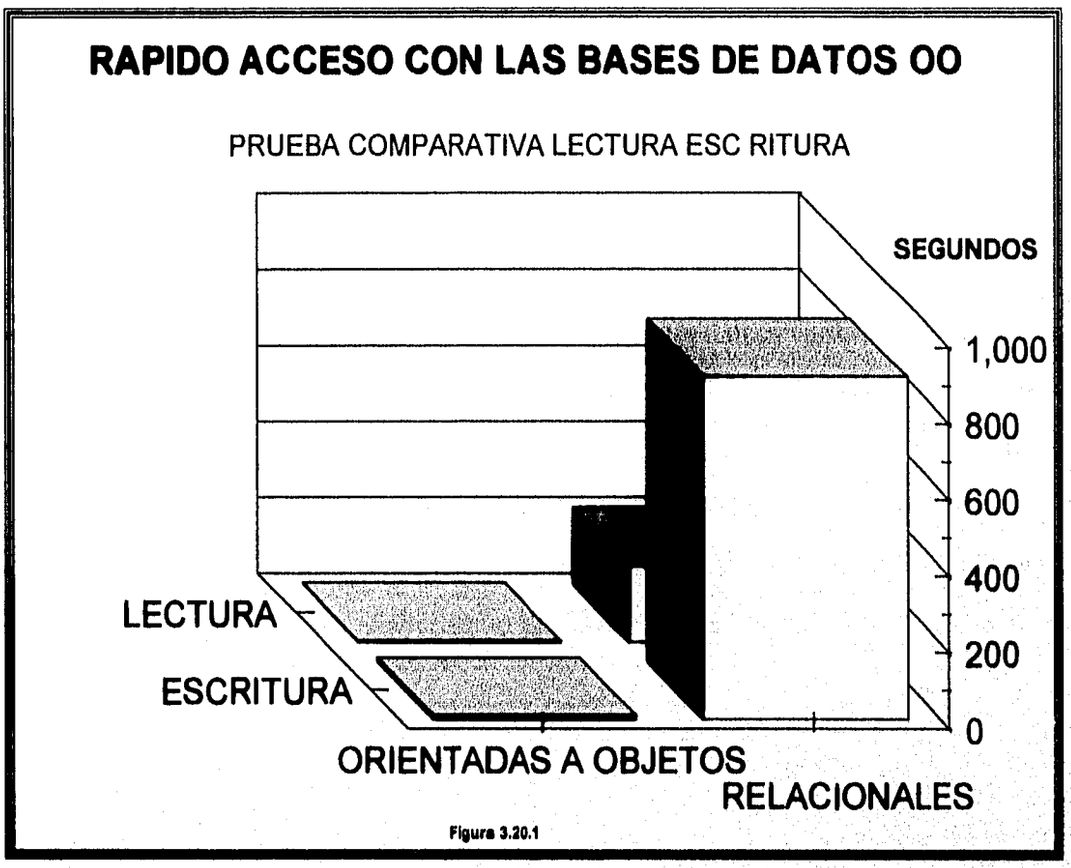


Figura 3.20.1

Mientras estos resultados son dramáticos, son también generalmente típicos del mejor desempeño que puede ser acrecentado con datos complejos. Los sistemas relacionales manejan transacciones simples más rápido que las bases de datos OO, y son mejores en relación a querys relacionales, pero **en el manejo de estructuras de objetos complejos, las bases de datos OO son considerablemente más rápidas.**

Un estudio hecho por Sun Micro Systems, estableció una marca para el tratamiento de información estructurada, y encontraba que cada DBMSOOs probado desempeñó todo, corrigiendo a los sistemas relacionales (Cattell, Object Data Managament: Addison-Wesley, 1991).

Los DBMSOO manejan bien las excepciones, ya que estos ofrecen la habilidad de manejar excepciones a estructuras de información normal, creando subclases. Por ejemplo, si usted tiene que almacenar direcciones foráneas en una base de datos que se designaron por direcciones locales, usted puede crear una subclase de la

clase dirección para manejar la información adicional. Eso le permite manejar el caso especial sin reestructurar todas las direcciones existentes.

Los primeros DBMSOOs fueron motivados a evolucionar, primero por el deseo de construir mejores tipos de DBMSOOs, en esta consideración se incluían: control de concurrencia, seguridad, integridad, y otros rasgos tradicionales de los DBMSs. Los lenguajes propietarios son usados con frecuencia. Al pretender una tendencia histórica los DBMSOO's primitivos definieron su propio lenguaje de definición de datos propio (DDL) y su lenguaje de manipulación de datos (DML) por ejemplo Servio's GemStone usa Opal, una versión propia de Smalltalk para ambos es DDL y DML. los productos Ontologic's Vbase usaron lenguaje c para definir su DDL y le llamaron TDL, para manipular los datos COP que equivale a un DML.

Resúmen de las Diferencias Entre las Bases de Datos por Relación y las Bases de Datos Orientadas a Objetos.

Características	Bases de Datos Relacionales	Bases de Datos Orientadas a Objetos
Objetivo Principal:	Independencia de los datos.	Encapsulamiento.
La B. D. Almacena	Unicamente datos.	Datos más métodos.
Datos Compartidos/ Encapsulado.	Los datos pueden compartirse entre varios procesos, ya que los datos están diseñados para cualquier tipo de uso.	Los datos solo pueden utilizarse por métodos de clases. Los datos están diseñados para su uso exclusivo por métodos particulares.
Datos Pasivos vs Objetos Activos.	Los datos son pasivos. Se pueden activar en forma automática ciertas operaciones limitadas al utilizarlos.	Los objetos son activos, ya que las solicitudes hacen que ellos ejecuten sus métodos, los cuales pueden ser muy complejos; por ejemplo, aquellos que utilicen reglas o máquinas de inferencias (sistemas expertos).
Cambios constantes.	Los procesos que utilizan los cambios cambian de manera constante.	Las Clases son diseñadas para su reutilización por lo que rara vez se modifican.
Independencia de Datos /clase.	Los datos pueden reorganizarse físicamente sin afectar su forma de uso.	Las clases se pueden reorganizar sin afectar su forma de uso.
Sencillez /Complejidad.	Los usuarios perciben los datos como columnas, renglones y tablas.	Las estructuras de datos pueden ser complejas, no obstante, para el usuario es transparente debido al encapsulado.
Tablas independientes/ Datos Ligados entre si.	Cada relación (tabla) es independiente. Los comandos **JOIN** relacionan los datos en las diversas tablas.	Los datos pueden estar ligados entre sí, de modo que los métodos de la clase logren un mejor rendimiento. Las tablas son sólo una de las estructuras de datos que se pueden utilizar. Los objetos binarios de gran tamaño (BLOB) se utilizan para aplicaciones multimedia y grandes flujos de bits sin estructura.

Redundancia.	La normalización de los datos se efectúa para ayudar a eliminar la redundancia en los datos (no ayuda en el caso de la redundancia en el desarrollo de la aplicación).	Con el encapsulado y la herencia se obtienen datos y métodos no redundantes. La herencia ayuda a reducir la redundancia en los métodos, mientras que la re utilización de las clases ayuda a reducir la redundancia general.
El SQL.	El SQL se utiliza en los DBMSR con buenos resultados. como DDL, DML y DQL.	Solicitudes OO. Las solicitudes provocan la ejecución de los métodos, los cuales utilizan el concepto de navegación para realizar búsquedas.
Rendimiento.	El rendimiento es una preocupación en el caso de estructuras altamente complejas.	Optimización de clases, los datos de un objeto se pueden ligar entre si y almacenar juntos, de modo que se pueda tener acceso a ellos desde una posición del mecanismo de acceso. Las BDOO ofrecen un rendimiento mejor que las BDR, para ciertas aplicaciones con datos complejos.
Modelo Conceptual.	El modelo de estructura de datos y acceso representado por tablas y comandos es distinto en cada fase del ciclo de vida del sistema; análisis, diseño y construcción, el diseño debe traducirse en tablas de relación y acceso al estilo SQL.	Modelo conceptual Consistente. Los modelos utilizados para análisis, diseño, construcción, acceso y estructura de la base de datos son similares. Los conceptos de la aplicación se representan de manera directa mediante clases en la BDOO. Cuanto más complejas sean las aplicaciones y su base de datos, más tiempo y dinero se ahorrarán en el desarrollo de aplicaciones.

3.21 LOS BENEFICIOS DE LOS DBMSOO

PRINCIPALES BENEFICIOS:

La capacidad de poder almacenar objetos en una base de datos nos permite compartirlos en un entorno distribuido. Una BDOO puede permitir que los objetos utilizados activamente sean cargados en memoria, y así minimizar o anticipar la necesidad de paginación de memoria virtual, lo cual es especialmente útil en sistemas de gran escala, la persistencia de los objetos nos permite que pueda existir una versión para cada objeto, lo cual nos es útil no solamente para cuestiones de comprobación y prueba sino también para muchas aplicaciones de diseño OO en las que el control de versiones es una exigencia funcional que debe tener el producto de desarrollo.

Las Bases de Datos OO ofrecen muchas de las ventajas que antigüamente sólo se encontraban en los sistemas expertos, con una BDOO, las relaciones entre objetos y sus limitaciones son administradas por el DBMSOO. Las reglas asociadas a un sistema experto son sustituidas fundamentalmente por el esquema, objeto y los métodos, como la mayoría de los actuales sistemas expertos no cuentan con un adecuado soporte de base de datos, las BDOO permiten obtener la funcionalidad de un sistema experto con mucho mejor rendimiento.

Las BDOO proporcionan ventajas en comparación con los actuales modelos de datos jerárquicos y relacionales. Posibilitan el soporte de aplicaciones complejas que no son soportadas perfectamente con el resto de los modelos.

Amplían la programabilidad y el rendimiento, mejoran el acceso mediante la navegación y simplifican el control de concurrencia, por lo cual se espera gran rendimiento con el procesamiento simétrico, o procesamiento masivamente paralelo, disminuyen los riesgos asociados con la integridad referencial y proporcionan un mejor modelo para el usuario final.

Las BDOO permiten por definición la inclusión de una mayor cantidad de código en la propia base de datos, este mayor conocimiento de la aplicación tiene grandes

ventajas potenciales para el propio DBMSOO, incluyendo la posibilidad de optimizar las consultas y controlar el acceso concurrente (multiusuario).

El concepto rendimiento, siempre importante en el desarrollo de sistemas de información, puede mejorarse significativamente utilizando un modelo de datos OO, en lugar de un modelo relacional. La mejora más significativa se puede esperar en aplicaciones grandes y complejamente interrelacionadas, el agrupamiento (clustering), o localización de objetos relacionados en proximidad cercana, puede realizarse a través de la jerarquía de clases o por medio de otras interrelaciones. El empleo de memoria caché (caching) en las BDOO saca su máximo provecho cuando existe una gran complejidad de datos al anticipar que el usuario o la aplicación puede recuperar una determinada variable de instancia (variable modelo) de la clase que retendrá en la memoria de almacenamiento "caché". Las BDOO pueden almacenar no sólo componentes complejos, sino también estructura de datos superiores, aunque los sistemas relacionales pueden utilizar un número considerado de tuplas, están limitados en tamaño; las BDOO no sufren de una degradación en el rendimiento ya que no tienen la necesidad de que los grandes objetos sean descompuestos y reensamblados por las aplicaciones, sin importar la complejidad de las propiedades de la aplicación.

Como los objetos contienen referencias directas a otros objetos, los conjuntos de datos complejos pueden ser ensamblados eficazmente utilizando estas referencias directas. La capacidad de buscar por referencias directas mejora significativamente el acceso por navegación. En contraste, los conjuntos de datos complejos (objetos) en las bases de datos relacionales deben ensamblarse por el programa de la aplicación, utilizando el lento proceso de unir tablas.

Para un programador, uno de los principales retos en el desarrollo de sistemas de bases de datos es el lenguaje de manipulación de datos (DML Data Manipulation Language) de la base de datos. En los DBMS Relacionales el DML difiere normalmente del lenguaje de programación utilizado para construir el resto de la aplicación (DDL Y SQL). Este contraste se debe a la falta de coincidencias de los

sistemas de tipo o a la diversidad de paradigmas de programación. Por lo que el programador debe aprender dos lenguajes, dos conjuntos de herramientas de programación, como por ejemplo los generadores de aplicación y los lenguajes de cuarta generación (4GL), que han surgido para producir el código de toda la aplicación, cerrando así el intervalo de falta de coincidencia entre el lenguaje de programación y el DML; pero la mayoría de estas herramientas comprometen el proceso de implantación del sistema y aplicación. Con las BDOO se elimina gran parte de este problema. El DML puede adaptarse para que una mayor parte de la aplicación pueda escribirse con el DML. O también puede ampliarse un lenguaje de programación OO como C++ para que éste sea el DML. Una mayor parte de la aplicación puede construirse con el DML con herramientas del DBMSOO. El movimiento a lo largo de la interfaz de programación entre la BD y la aplicación ocurre entonces con el único paradigma y sobre todo con un conjunto común de herramientas.

Las bibliotecas de clase ayudan al programador a desarrollar mas rápidamente, ya que fomentan la reutilización del código existente y ayudan a modificar el costo del mantenimiento.

La programación es más fácil, por que las estructuras de datos modelan el problema de una manera más fiel, es decir más semejante al mundo real.

La integridad de los datos es mejorada, ya que al estar encapsulados los datos y sus procedimientos en un único objeto, hace que la probabilidad de que un cambio a un objeto altere la integridad de otros objetos de la base de datos.

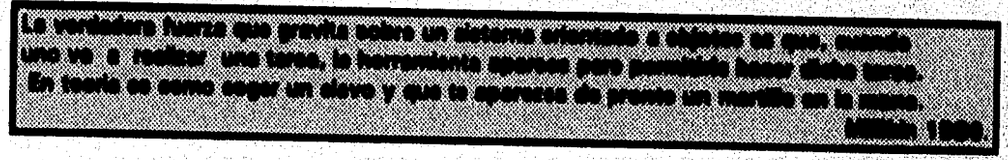
Las BDOO también mejoran el control de concurrencia en forma más simple que las Relacionales, ya que en éstas la aplicación necesita bloquear en forma explícita cada registro en cada tabla, por lo que los datos presentados en una vista, están formados por varios campos de diferentes tablas. La integridad una existencia clave en las bases de datos se soporta mejor en las BDOO ya que la aplicacion OO pude bloquear todos los datos relevantes de una operación.

La integridad referencial no está garantizada en una BD Relacional, en la que el borrado de un campo de un renglón de una tabla, puede dejar pendiente un puntero hacia el proveniente de otra tabla, a menos que la aplicación compruebe específicamente esta situación. La integridad referencial está mejor sustentada en una BDOO porque los punteros se mantienen y actualizan por el propio DBMSOO.

Las BDOO ofrecen una mejor metáfora de usuario que las otras BD Relacionales, la tupla o tabla con sus columnas y renglones aunque posibilita una estrategia de realización bien definida, no es una estructura de modelación intuitiva, especialmente fuera del dominio de los números. Los objetos ofrecen un esquema de modelación más natural y completo, soportando estructuras grandes y complejas como esta formado el mundo real.

CAPITULO IV

PERSPECTIVAS FUTURAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS



CAPITULO IV

PERSPECTIVAS FUTURAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS (BDOO)

4.1 EL TIEMPO DEL CAMBIO ES AHORA

La computadora es buena en ciertas tareas que la destreza humana no hace tan bién (rapidez y exactitud); sin embargo, el ser humano es magnifico en varias tareas que la computación hasta ahora no puede realizar (inventa, crea conceptos, se enamora, piensa en colonizar la luna, etc.). El reto de la Ingeniería en computación es forjar una asociación creativa que utilice lo mejor de ambos.

Las computadoras del futuro no serán como los robots que vemos en las películas. No tendrán las capacidades humanas del Schazeneger autómata. Serán más interesantes, de varias formas, puesto que utilizarán redes a nivel mundial de un inmenso ancho de banda y tendrán acceso a grandes cantidades de datos y enormes bibliotecas de complicado software orientado a objetos, absolutamente preciso en su funcionamiento. A mediados de los ochenta, las autoridades de las técnicas estructuradas afirmaban que era imposible construir sistemas de 50 millones de líneas de código, nuestros sistemas del futuro requiere software en los que sistemas de 50 millones de código sean comunes. En este aspecto son esenciales las técnicas orientadas a objetos, con encapsulado, polimorfismo, bases de datos orientadas a objetos, automatización de diseño y generadores de código.

Los sistemas actuales continúan creciendo en complejidad, la cual está aumentando en número de dimensiones, no sólo en lo que respecta a las exigencias de una funcionalidad amplia, sino también en términos de diferentes tipos de datos. La mayoría de los sistemas del futuro procesarán probablemente imagen, voz y video, además de texto y números. Como muestra la siguiente figura, la mejora del rendimiento y el aumento de la capacidad eran los retos del pasado, mientras que el desarrollo de una complejidad creciente es el reto del futuro.

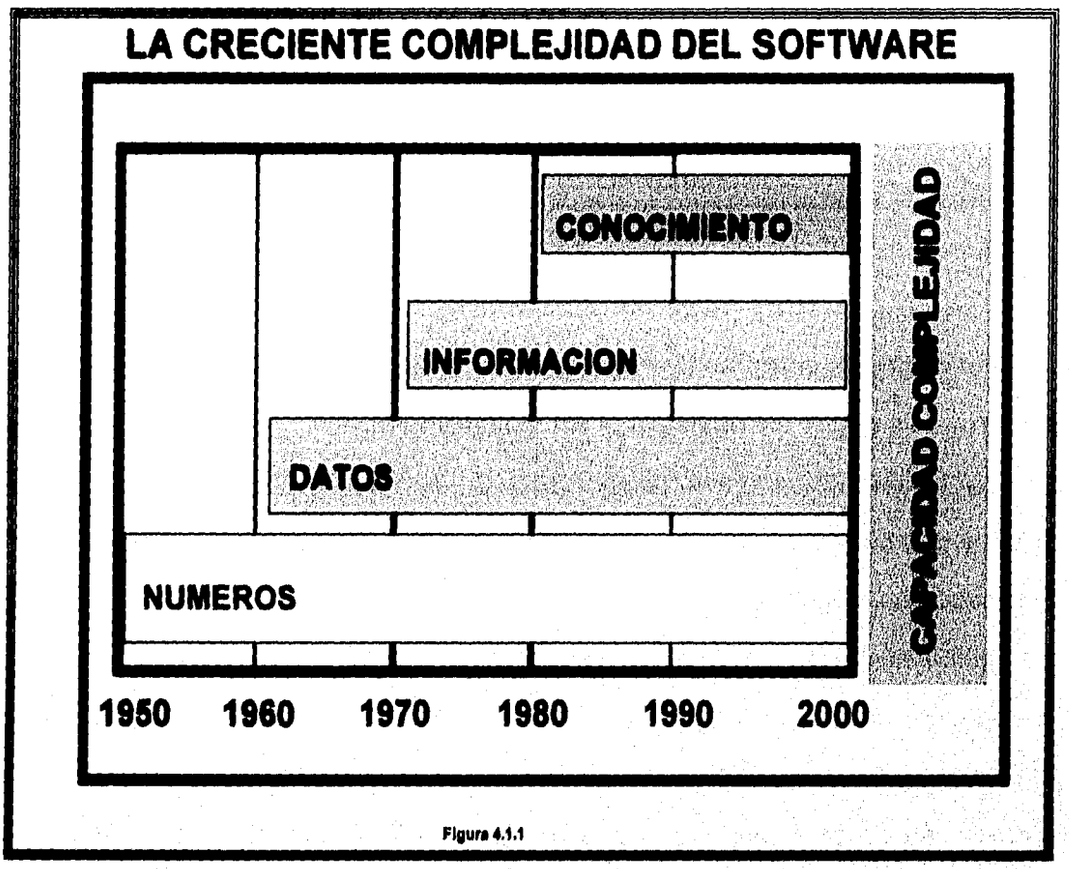


Figura 4.1.1

La informática ha sobrepasado el concepto de registro para adaptar tipos de datos más ricos y complejos. Areas de aplicación como por ejemplo, **CIM** Computer Integrated Manufacturing (Fabricación Integrada de Computadoras), **CAD** Computer Aided Design (Diseño asistido por Computadora), **CASE** Computer Aided Software Engineering (Ingeniería de Software Asistida por Computadora), **CAP** Computer Aided Publishing. (Edición Computarizada) ya agotaron la capacidad de programación actual.

A finales del siglo XX las aplicaciones necesitan satisfacer exigencias más sofisticadas, utilizar arquitecturas y estructuras de datos más complejas y ser accesibles a cada vez una más amplia base de usuarios, por lo tanto las bases de datos orientadas a objetos deberán coadyuvar a solucionar los problemas de complejidad; la orientación a objetos es un paradigma importante para los desafíos

en el desarrollo del software de los años 90's, promete un lanzamiento que aumentará gradualmente a lo largo de una amplia gama de tecnologías y saturará la próxima generación de arquitecturas de software. El paradigma de orientación a objetos mejorará el proceso del desarrollo de software y hará aparecer nuevas y mejores aplicaciones. El potencial ya está siendo demostrado y asumido por los creadores de software más avanzados que disponen de arquitecturas de objetos, de este modo aunque la orientación a objetos no está todavía incrustada completamente en las aplicaciones y herramientas del desarrollo de software que se utilizan actualmente, los fundamentos básicos para los entornos de desarrollo orientados a objetos están en manos de los principales creadores.

Las bases de datos orientadas a objetos, junto con el paradigma de orientación a objetos cambiarán la manera de pensar de los programadores, y aumentarán la velocidad de producción de la próxima generación de aplicaciones, también aumentará la capacidad de programación del usuario final, y la funcionalidad que pueda incorporarse a las aplicaciones. Los métodos orientados a objetos posibilitarán el que los usuarios accedan a los actuales tipos de plataformas de computación heterogéneas.

Las interfaces del futuro animarán a los usuarios a considerar todos los elementos visuales (íconos) como objetos para ser manipulados directamente con el "ratón" o teclado apoyados totalmente por un sistema de objetos subyacente.

4.2 EL SUBENIR DE DBMSOO ACTUALES

En el momento de realizar esta tesis existe muy poca información acerca de las aplicaciones desarrolladas con DBMSOO en México, sólo se encontró el producto VERSANT (en estudio) preparándose para el cambio en una área de Presupuesto de Petróleos Mexicanos, también se consultaron ejemplos en revistas de desarrollo con Ifel, Smalltalk, etc., más no DBMSOO.

La siguiente sección proveé una pequeña introducción a los principales productos de DBMSOO que actualmente están disponibles en el mercado. Mientras esta tesis es desarrollada, nuevos productos mejorados entran al mercado o bien nuevas versiones son liberadas con diversas mejoras. La siguiente lista de productos son descritos en orden histórico.

Servio: GemStone

Dirección: Servio Corporation.

1420 Harvor Bay Parkway

Suite 100, Alameda CA 94501

U.S.A.

Uno de los primeros DBMSOO multiusuarios en el mercado fue GemStone de Servio Corporation. GemStone está desarrollado en C y esta basado en un diseño sobre una arquitectura cliente/servidor con múltiples hilos de ejecución, GemStone tiene el lenguaje DDL/DML propietario llamado OPAL, que es esencial para DBMS orientados a versiones de Smalltalk. Todos los métodos del DBMSOO son escritos en OPAL. Las definiciones de clases persistentes son definidas con OPAL, así como todas las aplicaciones son desarrolladas con OPAL. GemStone soporta que el DDL/DML pueda integrarse con C, ADA, C++, Smalltalk (de ParcPlace y Digitalk) y cualquier lenguaje con C con llamadas a funciones externas. Smalltalk y GemStone soportan solamente la herencia simple.

GemStone puede compartir una interfaz con SYBASE, tiene un sistema de respaldo que permite uso contínuo las 24 horas del día, objetos definidos en C++, tiene su identidad propia y coexisten independientemente de la aplicación que los crea, luego pueden almacenarse y posteriormente reinvocarse por la misma aplicación u otras aplicaciones.

Su mayor virtud es el soporte para múltiples plataformas, estaciones de trabajo y servidores que van desde IBM, DIGITAL y SUN. Además puede implantarse en PC's IBM y compatibles, así como Maquintosh II. El producto incluye una herramienta gráfica para la definición y creación de clases llamada GemStone Visual Shema Designer. Además existe otra serie de aditamentos para el desarrollo disciplinado de aplicaciones. GemStone tiene más de 120 sistemas instalados internacionalmente.

Itasca:

Dirección: Itasca systems Inc.

2850 Metro Drive, suite 300

Mineapolis, MN 55425 U.S.A.

Este sistema fue derivado de Orion sistema desarrollado por MCC. Es implementado con Common-Lisp. El sistema soporta multicliente y multiservidores en donde cada nodo de la red puede ser dinámicamente configurado, no se requiere un server central con enlaces disponibles y corre en estaciones de trabajo de UNIX.

El modelo de datos de Itasca es una extensión de Common-Lisp. Los métodos son almacenados en la base de datos, soporta herencia múltiple, cuenta con estrategias para la solución de distintos conflictos, puede recibir mensajes en una gran variedad de caminos para los objetos ordinarios; estos mensajes regresan información sobre el esquema (nombres de los atributos, subclases, seguridad, etc.). Itasca no soporta atributos inversos, pero soporta objetos compuestos, Itasca soporta querys sencillos, subconjunto de una colección de objetos que pueden ser requeridos. Contiene un conjunto de herramientas gráficas que permiten editar las instancias y el esquema, las herramientas particulares de administración de la BD incluyen examen de autorizaciones, modificación de permisos, y afinación del funcionamiento de la base de datos.

Matisse:

Matisse es un DBMSOO Francés. Hoy en día es distribuido en los Estados Unidos de América por dos Proveedores:

ODB (Object Database)
238 Brodway
Cambridge, MA 02139
U.S.A.

Intellitic International SNC
14,rue du fort de Saint Cyr
BP 317 - Montigny le Bretonneaux
78054 Saint Quentin Yvelines
Cedex France.

Matisse es estructurado con un diccionario de base de datos con una semántica inferior a su modelo de datos, el cual es construido con estructuras de datos de alto nivel, otra característica de Matisse es que su DDL y DML son idénticos como consecuencia de su Meta modelo Orientado a objetos. Entre otras funciones Matisse efectúa respaldos incrementales, replicación de datos de forma transparente y recuperación dinámica de discos. Existe también control de concurrencia. La dinámica evolución de su esquema soporta versiones históricas de la administración de los datos. Además de soportar Dinámico Clustering e indexación. Adicionalmente contiene herramientas para desarrollo y administración, además de encriptación y compresión de datos.

O_2 DBMSOO

Dirección: O_2 Tecnology
7 rue, du Parc de Clagny
78000 Versaille
France

El DBMS O_2 es producido por la firma francesa Atair y es un producto comercial bastante conocido que incluye todo un ambiente completo de programación (Integrated Develoment Enviroment), su característica principal es la de ser un DBMSOO que soporta una arquitectura de: Múltiple Clientes/ Múltiples Servidores, su modelo de datos es similar al de GOM y fue desarrollado para soportar lenguaje C y C++ para definir operaciones un lenguaje derivado de C es usado llamado O_2C, soporta herencia múltiple, O_2 proveé un poderoso lenguaje de consulta que maneja un estilo SQL (select from where), el control de transacciones es optimizado en O_2, es posible la evolución del esquema resultando posible así la conversión de objetos si fuera necesario, su funcionamiento en tiempo real puede ser optimizando al indexar los clusters y la definición de trees. Existe soporte para incluir aplicaciones escritas en varios lenguajes, O_2 contiene un poderoso conjunto de browsers: browser esquema, browser clase, browser aplicación, browser función y browser para nombres persistentes. Todos los browser están basados en el sistema Windows OSF/motif contiene: Browsers: Depurador, Browser/Editor para los esquemas y los objetos, Tool Box (librería de clases y objetos), Herramienta para la generación de interfaces. Estaciones de trabajo, servidor(es), Trabaja en dos formas: de desarrollo y Ejecución, Multi-Lenguajes (C, Basic, LISP, C++,...) Atributos excepcionales y métodos, Administración de la persistencia, extensiones de clases, Verificación estática de tipos, Conflictos de herencia múltiple (resuelto por los usuarios).

Objectivity: Objectivity/DB

Dirección: El camio Real, 4th floor

Menlo Park, CA 940025 U.S.A.

El DBMSOO Objectivity/DB es el producto principal de la compañía Objectivity Inc. Este software está enfocado a las necesidades de los ingenieros, usuarios científicos y técnicos. Objectivity/DB puede instalarse en una amplia variedad de plataformas y dominios de ingeniería, incluyendo redes heterogéneas que combinan estaciones de trabajo Sun y Digital, proveé de una manera transparente todo el poder de los sistemas distribuidos, múltiples bases de datos pueden estar localizadas en diferentes redes heterogéneas. Objectivity/DB es uno de los primeros DBMSOO asociado con uno de los mejores distribuidores como lo es Digital Equipement Corporation, el estudio de MIT no incluye desarrollos europeos, por lo que en la tabla no aparece.

El diseño de Objectivity/DB escogió a C++ como su modelo de datos en donde es soportada la herencia múltiple, entre otras cosas, ya que soporta también objetos complejos y relaciones binarias llamadas asociaciones por cardinalidad, adicionalmente soporta arreglos de tamaño variable con lo que se pueden efectuar dinámicos cambios. Objectivity/DB soporta administración de configuración, soporte para control de versiones, mecanismos para el soporte de transacciones largas y recuperación, así como protección vía paswords, etc.

La optimizacion del funcionamiento es posible vía clustering.

Una adicional mejora es la inclusión de hipertexto gráfico, proveído en la base de datos, además de contener herramientas browsers y debuggers. Para satisfacer de manera acorde las necesidades de los ingenieros usuarios, Objectivy maneja los datos a nivel de objetos no a nivel de archivos, por lo que la estructura de datos es rica para aplicaciones de ingeniería compuestas de objetos complejos. El cliente más renombrado de Objectivity/DB es el fabricante de aeroplanos Boeing, en Seatle. USA.

Versant: Versant ODBMS

Versant es un ODBMS que incluye todas la futuras necesidades para escalar en cuanto a producción bases de datos distribuidas, además de un entorno de workgroup heterogéneo, es un modelo multicliente/multiservidor que desarrolló Versant Obvject Tecnology, está disponible para casi todas las plataformas, ya que es el producto OO que soporta más estándares, incluyendo X-Windows, OSF/Motif, ANSI C, AT&T C++2.0, Smalltalk-80 y TCP/IP.

Versant permite aplicaciones multiusuarios, una aplicación puede actualizar datos desde varios servidores en una sola transacción, eliminando la necesidad de grandes bases de datos centralizadas que permite distribuir los datos donde son más frecuentemente utilizados. Los objetos pueden extraerse en préstamo de un grupo a una base de datos local para transacciones locales. VERSANT soporta Two-phase-commits y recuperación de rollback garantizando actualizaciones atómicas. Acompañan al manejador varios productos y herramientas para su desarrollo, entre ellas: Versant View, Versant Schema Designer, Versant C++ y Versant 4GL; veamos algunas de las características más importantes:

- Versan emplea el modelo de datos orientados a objetos, ya que permite la implementación de objetos.

Las características generales de Versan son:

- Personalización para definir tipos de datos complejos.
- Encapsulación de datos y código.
- Asociación de datos enlazados con arreglos de uniones.
- Código reusable.
- Polimorfismo.
- Identificación única de objetos.
- Tipos extensibles de datos.
- Tipos parametrisados.
- Objetos embebidos (inmersos).

- Containers y colecciones de objetos.

La base de datos de Versant incluye las siguientes características:

- Persistencia al guardar los datos.
- Acceso concurrente a múltiples usuarios.
- Acceso concurrente por procesos múltiples en aplicaciones sencillas.
- Administración de transacciones.
- Recuperación desde fallas del sistema.
- Navegación y búsquedas por medio de consultas condicionales.
- Conexiones a bases de datos remotas.
- Versiones de datos.

El soporte de la base de datos de Versant para el acceso incluye las siguientes características:

Interfases con múltiples lenguajes.

Plataformas heterogéneas.

Múltiples compiladores.

Distribución de datos incluyendo la habilidad de migrar objetos.

Utilización del hardware con una arquitectura cliente/Servidor.

Procesamiento de consultas en Servers.

Administración dinámica del esquema de la base de datos.

Diferentes niveles de seguridad de objetos.

Cambio de objeto en el cliente y cambio de página en el server.

Grupos de instancias de una clase.

Índices para la optimizacion de Querys.

Replicación de datos en numerosas bases de datos.

Object Design: Object Store

Dirección: Object Design Inc.
1 New England Executive Park
Burlington, MA 01803, U.S.A.

Object Store es un DBMSOO para estaciones de trabajo SUN3 Sun Sparc. Existen versiones para Microsoft Windows 3.0 y otras estaciones de trabajo en UNIX. Object Store y sus herramientas de trabajo están basados en C++. El preprocesador proporciona una interfaz de desarrollo que permite comprimir código de C++ en enunciados optimizados. Además, el DBMS Object Store permite tipos parametrizados, los cuales son comúnmente utilizados para definir clases que contienen e incorporan código de C++ y que permiten desarrollo de código C++ reutilizable. Object Store exhibe una arquitectura de sistemas multicliente/multiservidor, es implementado con C y C++, una de sus principales características es el empleo del mapeo de la memoria virtual. El modelo de datos de Object Store es una extensión del lenguaje de programación C++. Los tipos de objetos son llamados clases en C++ los miembros de los atributos y los tipos de operaciones asociadas a las funciones. En Object Store los miembros de las funciones son almacenados en C++ por medio de archivos binarios, para ayudar a la funcionalidad de la base de datos. Existe una colección de querys soportada por Object Store, además de soportar relaciones binarias y cardinalidad de tipo 1:1, 1:n, y n:m en forma de atributos inversos la integridad de estos atributos es automáticamente administrada y mantenida por el sistema los objetos pueden ser borrados explícitamente por el usuario de la base de datos. No contiene la recolección automática de basura como otros DBMSOO. Si un peligro es detectado, automáticamente Object Store proporciona dos mecanismos para manejar esto, la primera reacción es efectuar (raising) una excepción, la segunda es asignar una

referencia Nula a esa referencia. El usuario puede escoger entre estas dos alternativas.

La persistencia en la definición de clases es totalmente ortogonal. Object Store proveé múltiple granularidad asegurando los objetos en la base de datos, los niveles de granularidad son segmentados y paginados en grupos de objetos llamados colecciones Una de las dos fases del protocolo de seguridad garantiza la seriabilidad de las transacciones, a continuación la detección automática del deadlock, y la última transacción involucrada es restaurada por default. Al restaurar, es inicializada la unidad de transacción de terminales satisfactoriamente o se especifica el número de transacciones excedidas. El funcionamiento del run-time pude ser eficientado incrementando el clustering definido por el usuario. Objetos muy grandes pueden ser soportados en múltiples páginas, éstos pueden ser usados para almacenar información de multimedia. Un diseñador gráfico de esquemas y un browser interactivo están disponibles en Object Store.

Ontos

Dirección: Ontologic Inc.
3 Burlington Woods
Burlington MA 01803, U.S.A.

Ontos que ofrece la compañía Ontologic, Inc. exhibe una arquitectura de sistemas multiclientes/multiservidores, es implementado con C++, su principal característica es la de incluir una interfaz completamente transparente con C++ (Esto es transparentemente activa, y objetos en la memoria como se requiere dinámicamente), además permite activar objetos desde el servidor de la base de datos y proporciona una interfaz muy versátil con otros productos y herramientas de Ontos como Ontos SQL, Ontos Studio, Ontos Smorthand y Ontos Ptech. Ontos está totalmente basado en el modelo cliente/servidor, el protocolo de transacciones distribuidas soporta "**two-phase-commits**" atómicos cada proceso cliente trabaja con una base de datos lógica que es distribuida sobre diversos distribuidores, un servidor es distinguido como el servidor primario y canaliza y controla el tráfico para los demás clientes, la distribución de los datos es transparente para el usuario. Ontos soporta la coexistencia de diversos administradores de almacenamiento.

El modelo de datos que maneja Ontos es C++, los métodos son almacenados en ordinarios archivos binarios y no en la base de datos, éstos son enlazados dinámicamente con los datos del objeto mientras éste es accesado. Los métodos pueden ser invocados clásicamente por una llamada en la compilación o para facilitar la interpretación de las cadenas o nombres en la run time, las operaciones de Ontos son llamadas a las librerías.

Ontos soporta atributos inversos y la integridad es mantenida automáticamente. La cardinalidad de los posibles tipos de relaciones binarias son soportadas por Ontos. Ontos emplea identificadores de objetos lógicos, los objetos son borrados explícitamente. No soporta la recolección automática de basura.

Para optimizar el funcionamiento Ontos permite Clustering, indexaciones, y manejo de caché. La persistencia de los objetos puede tener dos estados: deactivated o activated. Deactivated almacena los objetos en disco después de la activación, los objetos son transferidos a memoria principal y pueden ser tratados como los objetos comunes de C++. La representación de disco con la de memoria principal difiere.

Activated: Permite al programador el control automático y explícito de la persistencia mediante referencias.

Ontos soporta clustering físico de objetos. Para mejorar el rendimiento puede ser definido un cloustering lógico, además de contener herramientas interactivas para editar el esquema, y browser de objetos. Una herramienta es incluida para la agregación de objetos y para la creación dinámica de nuevas clases, además de contar con un poderoso lenguaje de 4GL para construir rápidamente reportes. Un poderoso cargador de datos esta disponible para importar datos de otros sistemas, además de existir las herramientas necesarias para administrar la base de datos, vía estas herramientas es posible modificar el clustering físico de los objetos.

Los lenguajes soportados por Ontos son: C, C++, Smalltalk y Ada para datos de multimedia, Ontos proveé soporte para Objetos de gran tamaño.

El producto Ontos está disponible para Sun, ·4 y SPARC, las series Apollo 3000 y 4000 y múltiples equipos en OS/2.

Open ODB

Dirección: Open ODB
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304
U.S.A.

Open ODB es resultado de las investigaciones de Iris que realizó HP en sus laboratorios de Palo Alto California, la arquitectura de Open ODB es cliente/servidor. El servidor de objetos administra el almacenamiento por arriba de los DBMS Relacionales . El modelo de datos de Open ODB está basado en el modelo funcional de los datos que es una consecuencia del modelado de funciones el modelo de datos y el lenguaje de consultas de Open ODB es llamado OSQL.

La herencia sencilla y múltiple es soportada. Durante el ciclo de vida de los objetos se les pueden asociar dinámicamente diferentes tipos, es posible que un objeto sea distribuido en diferentes tipos al mismo tiempo.

Open ODB proveé registros para control de transacciones durante las secciones. OSQL contiene muchos conceptos de autorización ortogonal. Existen dos reglas que pueden garantizar el funcionamiento del grupo de usuarios CALL y UPDATE. El clustering y la indexación son soportados por Open ODB para optimizar el funcionamiento, además de contener un browser gráfico interactivo que permite al usuario modificar el esquema y el contenido de la base de datos, el cual sirve para localizar código y reutilizarlo en las clases.

Statice

Dirección: Symbolics Inc.
8 New England Executive Park, East
Burlington, MA 01803 . U.S.A.

El DMBSOO Statice presenta una arquitectura cliente/servidor y esta basado en Common-Lisp.

El modelo de datos es similar a Flavor del modelo de datos de Orion aunque también esta basado en lisp, incluye herencia múltiple. El acceso asociativo puede ser usado, requiere el borrado explícito de los objetos, si un objeto es borrado todas las referencias a ese objeto serán también borradas, para incrementar el funcionamiento permite indexación de B-tree y el clustering por los valores de los atributos es posible.

UniSQL

Dirección: UniSQL, Inc.
9390 Research Bldv.
Austin TX 78759. U.S.A.

El DMBSROO UniSQL presenta algunas diferencias con respecto a los anteriores DBMSOO descritos, ya que no esta clasificado como un DMBSOO Puro sino como un DBMSR extendido, por un lado contiene todo el poder de los DBMS- Relacionales con algunas mejoras y por otro el usuario puede incorporar sus necesidades orientadas a objetos, para ello soporta tablas anidadas, es decir, el campo de una tabla puede contener otra tabla y los atributos o campos pueden contener los procedimientos (métodos) y los apuntadores, además soporta herencia múltiple en este aspecto Uni SQL es muy similar al prototipo investigado por POSTGRES y su producto comercial llamado Miro. La documentación de UniSQL es suficiente.

ORACLE- SEDONA

En lo que fue el primer evento de bases de datos en México, en el Word Trade Center de la Cd. de México, durante la primera reunión "Data Base Cliente/Servidor '95" y "Unix Open '95" ORACLE de México anunció oficialmente su entrada al campo de la orientación a objetos con SEDONA, la cual correrá bajo ORACLE 7 así como en su próxima generación, ORACLE 8, versión en la que se espera incluir una capa de conversión a objeto corriendo por encima del motor relacional. Según los responsables de ORACLE, la clave de SEDONA se encuentra en el repositorio que está diseñado para soportar múltiples funciones. De hecho, los responsables de la compañía lo definen como algo que podría actuar como una fábrica en donde las piezas y los modelos de objeto podrían ser ensamblados y guardados hasta que recibiera los mensajes de las aplicaciones solicitadas. SEDONA podría en teoría, proveer integración a la vez que construir una capa de conversión de relacional a objeto. Aún así, los analistas del mercado estadounidense apuntan que ORACLE debería tener cuidado en no intentar posicionar una plataforma de objeto que no se adecúe perfectamente con otros fabricantes. De hecho, la compañía esta solicitando ayuda de otros proveedores (Sun Microsystems, por ejemplo, esta apoyando a ORACLE con el Object Mediator) con el objeto de proveer las partes claves de SEDONA.

4.3 TECNOLOGIA INFORMATICA DEL SIGLO XXI

Las soluciones de cómputo y manejo de información para el siglo XXI, estarán sustentadas en aspectos como computación cooperativa, extracción de datos, procesamiento paralelo, bibliotecas digitales, modelos de clases, programación visual, multimedia y por supuesto lo anterior girando entorno a la orientación a objetos.

El futuro inmediato demanda bases de datos móviles, servidores que logren comunicarse con portátiles y sistemas digitales inalámbricos dentro de las empresas

y DBMS que soporten multimedia como lo hacen ya algunos DBMSOO. Para el procesamiento simétrico, la orientación a objetos es la mejor opción que existe actualmente para el desarrollo de bases de datos con capacidad de aprovechar computadoras que soporten el multiprocesamiento o procesamiento masivamente paralelo.

Veamos algunos aspectos de importancia en el futuro de las BDOO:

- El concepto de objeto tendrá que redefinirse debido a la convergencia de ideas procedentes de la inteligencia artificial.
- La orientación a objetos será una característica estándar de los lenguajes procedimentales (3GL) y surgirán nuevos lenguajes declarativos (4GL) orientados a objetos que reemplazarán a los ya existentes. Para la implementación del software, C++ seguirá siendo la opción preferida, aunque luego competirá en este respecto con Object COBOL, Smalltalk y CLOS serán los lenguajes preferidos para la elaboración de prototipos y entrenamiento.
- Los sistemas pequeños y los que involucran objetos complejos utilizarán bases de datos orientadas a objetos, mientras que los grandes sistemas comerciales que manejan grandes volúmenes de registros se seguirán construyendo con productos relacionales. En todo caso los sistemas darán soporte a estructuras organizacionables adaptables y serán orientados a objetos y basados en reglas.
- Se consolidará el movimiento de los sistemas abiertos a través de su adopción de la orientación a objetos y su filosofía de producir sobre un estándar más que estandarizar sobre un producto. El desarrollo de software se tomará cada vez más en un proceso industrial basado en la producción en masa de componentes.
- Máquinas más rápidas, posiblemente paralelas, promoverán los lenguajes orientados a objetos en la medida que ellos utilizan ligadura dinámica y recolección de basura. Los grandes sistemas de hardware se convertirán en servidores de redes.

¿Qué implica cambiar a esta nueva tecnología?

Existe una gran dificultad para migrar de los anteriores modelos de datos al modelo orientado a objetos. Yo estimo que aproximadamente dentro de unos 5 años, es decir a principio del 2000, en México se empleará el modelo orientado a objetos, como el modelo de datos fundamental para cualquier desarrollo de software.

4.4 TEMAS TÉCNICOS EN EVOLUCIÓN

La tecnología de DBMSOO aún es joven y está en un acelerado proceso de maduración, no existe aún la suficiente experiencia en el desarrollo de aplicaciones reales, cada etapa ha sido un paso en el proceso de aprendizaje, incluso para los proveedores de productos. No existen estándares generalmente aceptados y puestos en práctica por los miembros de la comunidad, varias iniciativas existen en este sentido y empiezan a concretarse algunos resultados.

Otros aspectos sobre los cuales los especialistas continúan trabajando son:

- **El SQL orientado a objetos**
- Un tema de gran debate es la definición e implantación de un lenguaje de consulta orientado a objetos, similar a un Object SQL que establezca la armonía entre el principio de encapsulamiento y la necesidad de conocer el contenido de los objetos por su valor y no por su identificador, sin violar el principio de la encapsulación.
- **metodologías para el análisis y diseño del modelo conceptual de una BDOO** Empiezan a proponerse y utilizarse nuevas metodologías para el desarrollo de aplicaciones complejas, ya que un buen diseño determina la eficiencia del sistema final.
- **el desarrolo de herramientas que exploten multimedia con DBMSOO.**

BASES DE DATOS INTELIGENTES (BDI)

Las BDI representan la evolución e integración de las BD tradicionales (jerárquicas, red y relacionales) empleando para ello las bases de datos OO[@]; integran además, las técnicas avanzadas para el almacenamiento y recuperación de información, la noción de hypermedia, los avances logrados en Inteligencia artificial particularmente en Sistemas Expertos. Su principal objetivo es permitir al usuario manipular el contenido de su base de datos de manera transparente, amigable y visual, ofreciéndole herramientas para extraer conocimientos, a partir del análisis del contenido de los datos, interrelacionar información provenientes de diferentes fuentes apoyado para ello en multimedia. Usar conocimientos y hacer inferencias para facilitar la recuperación y visualización de la información apoyando la toma de decisiones.

Los sistemas inteligentes son aquellos que están más relacionados con el **qué** desea su aplicación que con el **cómo** obtenerlo. Las BDI al integrar todas estás tecnologías, ofrecen una alternativa para representar y manipular la semántica ("Se entiende por semántica de una aplicación, la caracterización de los objetos que manipula, de sus propiedades estructurales y de su comportamiento") de las aplicaciones avanzadas, tales como CAD, CAM, CASE, Automatización de oficinas, etc. Las BDI proporcionan un marco común para almacenar y acceder la información para análisis y la toma de decisiones. Los objetivos que una BDI debe satisfacer son los siguientes:

- Mejorar la generación actual de lenguajes de bases de datos dirigidos al usuario ocasional o ingenuo y a los desarrolladores de aplicaciones.
- Herramientas para programadores y no programadores para construir interfaces y objetos gráficos como Query By Example-QBE, Structured Query Language-SQL y los lenguajes de 4GL.

[@] Para mayor información ver Ullman, J. D. "Principles of Database and Knowledge- Base Systems", Computer-Science Press 1988.Josep J.V. "Object Oriented Databases: Design and Implementation" Proceeding of the IEEE, Vol 79 No1 January 1991

La arquitectura de una base de datos inteligente esta compuesta por:

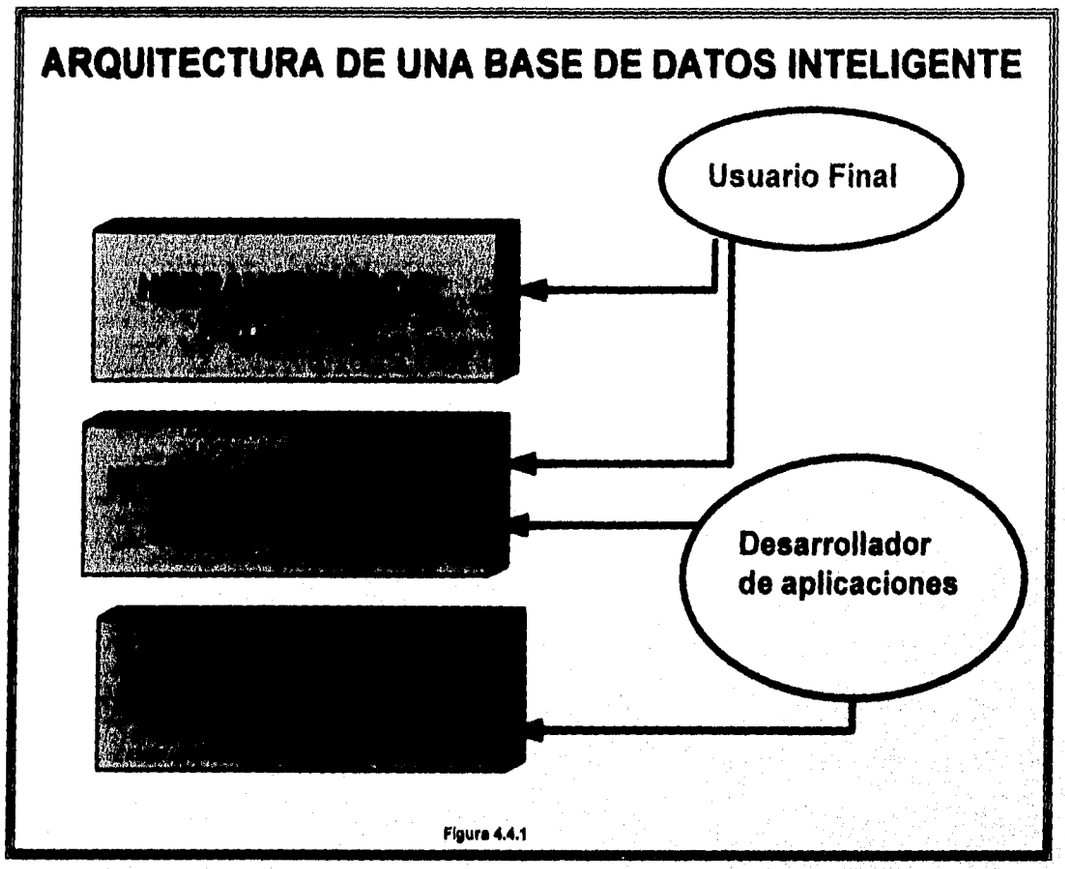


Figura 4.4.1

SISTEMAS OPERATIVOS ORIENTADOS A OBJETOS

Un sistema operativo es un conjunto de programas que administran el funcionamiento de la computadora, ayudando así éste a explotar los recursos informáticos (software, hardware y comunicaciones) NEXT STEP. Este nuevo sistema operativo es una interface gráfica con conectividad extensiva, acceso a bases de datos y vínculos de información, proporciona a estaciones de trabajo y máquinas basadas en Intel, diversas cualidades de interface gráfica, conectividad extensiva, acceso a bases de datos, vínculos de información, así como una arquitectura poderosa orientada a objetos.

La Arquitectura Orientada a Objetos en los Años 90's

El objetivo final de las aplicaciones orientadas a objetos es una arquitectura de objetos integrada con lenguajes, herramientas, ODBMS, Sistemas operativos; todos ellos trabajando juntos.

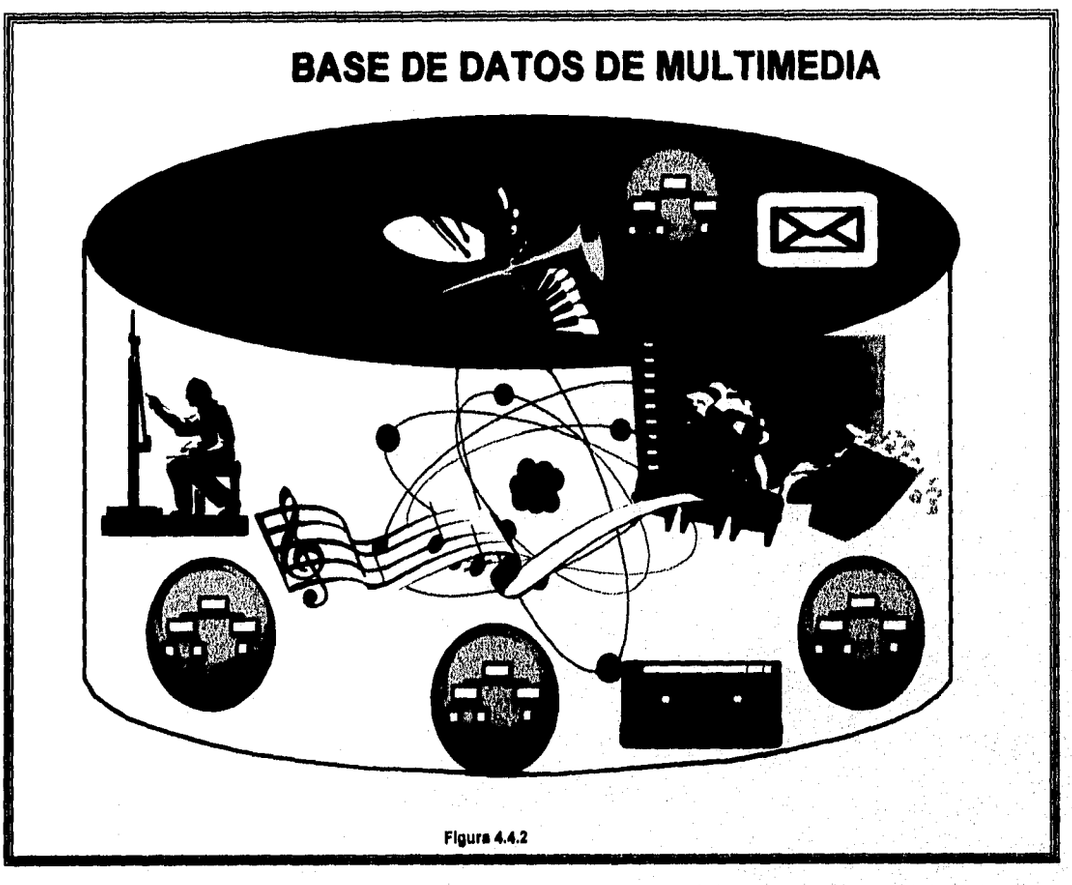


Figura 4.4.2

4.5 EMPRESAS QUE YA ESTAN UTILIZANDO LA TECNOLOGIA

El Maintenance Managament System

Con sus mas de 20,000 programadores, EDS es el mayor integrador de sistemas del mundo. Intrigados por la promesa de la orientación a objetos de duplicar la productividad, los directivos de la corporación decidieron realizar un experimento para tasar el potencial de la nueva tecnología. Para ello, encargaron a otra empresa, la Servio Corporation, el desarrollo de una versión orientada a objetos del Maintenance Managament System (MMS), una aplicación de mantenimiento industrial al servicio de General Motors. El MMS existente constaba de 256,000 líneas de código PL/1 y ocupaba 13.6 Mb de RAM; su desarrollo requirió 152 hombres-mes y tomó un total de 19 meses. El nuevo sistema consta de 22,000 líneas de código Smalltalk-80, ocupa 1.1 Mb de RAM, requirió 10.4 hombres-mes y se desarrolló en 3.5 meses.

El Customer Management System

El customer Management System (CMS) de la Broclyn Union Gas fue uno de los proyectos de gran escala que empleó la tecnología de objetos. Iniciando en 1984, el proyecto consistió en substituir la versión instalada de sistema desde hacía 13 años, la cual resultaba muy rígida frente a los constantes cambios en los requerimientos. El CMS era utilizado por el 80% de los empleados de la compañía para atender a un millón de clientes y obtener diversos servicios de procesamiento de datos tales como facturación, lectura de medidores y ordenes de servicio. El sistema se construyó entre 1987 y 1990 empleando un preprocesador de objetos que producía código PL/I, desarrollado en equipo con la Andersen Consulting. El producto obtenido contiene más de 10,000 módulos, 400 programas en línea, 118 programas batch, 150 diálogos en línea y 251 reportes. En total consta de 900,000

líneas de código fuente, 40% menos que el predecesor. La Brocklin y la Andersen hacen las siguientes observaciones con relación al proyecto:

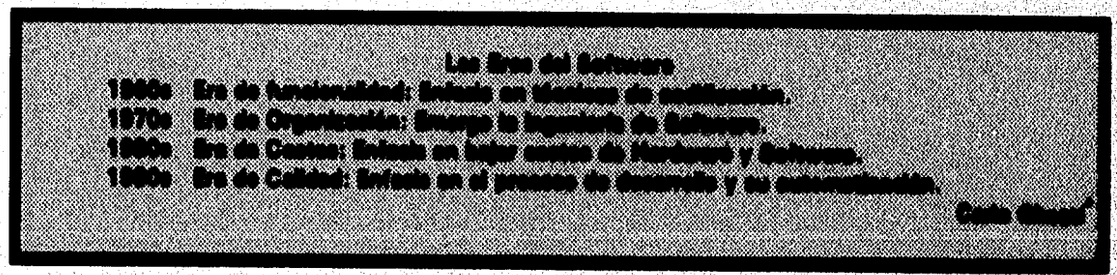
- Un sistema orientado a objetos es un modelo de la empresa.
- Los objetos saben defender sus datos mejor que nadie.
- La tecnología de objetos requiere nuevas metodologías.
- La tecnología de objetos permite a los sistemas cambiar más rápidamente que las empresas.

El Macintosh Support System

Toda empresa necesita herramientas para integrar su información en un documento gráfico que permita sustentar sus decisiones estratégicas y tácticas. Tales sistemas reciben el nombre genérico de "sistemas de sustento de decisiones" o "cuadros de mando". Apple computer fue pionera de estos productos en computadora personal a través de su Macintosh Desicion Support System (MacDSS), desarrollado en cooperación con KPMG, quien aportó su experiencia en el campo de las finanzas. El producto consta de 200,000 líneas de código y fue construido en Object Pascal con interfaz MacApp. James Joaquin, líder del proyecto pudo afirmar:

"La utilización de diseño y programación orientada a objetos es incuestionablemente la razón primaria de la terminación satisfactoria del proyecto MacDSS".

CONCLUSIONES



* C. Ghezzi, M. Jazayeri, D. Fundamentals of Software Engineering, Prentice Hall, 1991

CONCLUSIONES

Al principio de este trabajo se manifestó la necesidad de proporcionar soluciones a la crisis del software, particularmente esta tesis estudia la revolución de las bases de datos; es decir, el cambio radical en la forma de pensar acerca de cómo solucionar algunos de los problemas más relevantes en la implantación de sistemas de bases de datos y así buscar una solucion a la crisís del software.

La finalidad de este trabajo fue el proporcionar una introducción a la terminología de la orientación a objetos, donde se mostró la jerga que obscurece a la tecnología. Además se estudiaron algunas ventajas y desventajas de las bases de datos orientadas a objetos (bases de objetos), comparándolas con las relacionales; finalmente se mencionaron algunos productos de DBMSOO existentes en el mercado y se efectuó una proyección corta del futuro inmediato de las bases de datos orientadas a objetos, por lo que podemos concluir que:

1.- El método de orientación a objetos promete un nivel de abstracción mayor que el método soportado por los conceptos tradicionales de procedimientos y datos. En la orientación a objetos los procedimientos y datos se agrupan para formar una entidad denominada objeto. A diferencia de los datos pasivos y de los sistemas tradicionales, los objetos pueden actuar. Las acciones son desencadenadas por los mensajes enviados a un objeto y procesadas a continuación por procedimientos internos denominados métodos, por lo cual autores como James Martin han llamado a las bases de datos Orientadas a Objetos, "Bases de Datos Inteligentes".

2.- Los objetos pueden ser bastantes similares en sus posibilidades, la descripción de un conjunto de objetos similares es una clase. Un modelo o instancia de una clase es un objeto real descrito por una clase. La abstracción es estimulada mediante el empleo de jerarquías de clases. Para reutilizar los métodos definidos previamente, las jerarquías de clase cuentan con la herencia exclusiva de los lenguajes orientados a objetos, la cual facilita el desarrollo y mantenimiento de programas.

3.- Asimilar las ideas orientadas a objetos significa aprender las diferencias, así como las semejanzas entre el método OO y la programación tradicional.

Las bases de datos orientadas a Objetos producen sistemas de base de datos más fácil de implantar, sobre todo de mantener, aplicaciones más completas, más fáciles de utilizar y más flexibles, ya que el usuario puede acceder y adaptar elementos de la biblioteca de clases subyacente. El costo es significativamente menor.

4.- Las bases de datos orientadas a objetos son importantes hoy en día debido a la creciente necesidad de soportar estructuras cada vez más complejas como son: multimedia, computación del usuario final, programación visual, etc.

5.- Las bases de datos orientadas a objetos facilitan la tarea de integrar aplicaciones dispersas en distintas bases de datos con distintos formatos.

6.- Las bases de datos orientadas a objetos unifican los modelos, ya que a lo largo de todo el ciclo de vida del sistema (análisis, diseño, instrumentación, mantenimiento) se emplea el modelo orientado a objetos.

7.- Las BDOO tratan de que no nos interese más la estructura de las llaves que con el modelo relacional son fundamentales para el diseño y operación de sistemas.

8.- Las BDOO ofrecen mucho mejor rendimiento de la computadora que las relacionales para aplicaciones de estructuras complejas con gran cantidad de datos, **sin embargo para algunas aplicaciones tradicionales, las relacionales superan en rendimiento a las orientadas a objetos.**

9.- las BDOO coexistirán junto con las BD Relacionales durante algunos años, mientras exista un cambio en la cultura informática, y empiecen a migrarse los sistemas relacionales a un modelo OO más maduro, además de que el costo que representa para una empresa cambiar la forma de administrar sus BD, ya que si múltiples aplicaciones comparten una base de datos empresarial, es difícil y caro cambiarla; por lo que **las BDOO han incorporado el SQL para que todos los usuarios de SQL, tengan continuidad en sus tareas rutinarias, además las BDOO incorporan programas convertidores totalmente automáticos para migrar el modelo relacional a uno orientado a objetos.**

10.- Algunos sistemas OO son construidos con bases de datos relacionales extendidas, tratando de que el usuario preserve sus inversiones existentes a costa de la eficiencia.

11.- Recordemos que las bases de datos Orientadas a Objetos son bases de datos Inteligentes, ya que están compuestas de objetos activos, y estas reflejan de manera directa el modelo orientado a objetos de la empresa. Las aplicaciones de bases de datos orientadas a objetos futuras, probablemente proporcionarán entornos completos de programación visual.

12.- En cuanto al futuro inmediato de las BASES DE DATOS ORIENTADAS A OBJETOS, irán madurando e introduciendose al mercado con una fuerza mayor, cuando exista un estándar en cuanto a DBMSOO.

Este trabajo se ha realizado con la finalidad de acrecentar el acervo cultural de la Universidad Nacional Autónoma de México, que me brindó la oportunidad de estar en sus aulas para este propósito; a través de este trabajo he plasmado un trabajo de investigación acerca de temas relevantes en la computación, como son las bases de datos orientadas a objetos.

Muchas personas han contribuido a los avances del saber humano, para ellos se han empleado grandes esfuerzos, cuya recompensa ha sido la satisfacción de saber que la labor desarrollada sirva para ayudar a la formación de nuevas generaciones, de esta manera, deseo humildemente seguir el ejemplo de todos aquellos que fundaron las bases del conocimiento de la ciencias de la computación, legando a la humanidad este trabajo.

GLOSARIO

GLOSARIO

Este apéndice proporciona una breve definición de los términos técnicos y siglas que se emplean en la tesis. La definición completa de un término y su relación con las Bases de Datos Orientadas a Objetos se puede encontrar en el texto principal de la tesis.

- **Términos**
- **Siglas**

4GL: Fourt-Generation-Language, Lenguaje de cuarta generación. Lenguaje de computadora más especializado y avanzado que los de tercera generación. Utilizados para el desarrollo de aplicaciones actuales con un manejo ampliado de presentación y entradas y salidas.

5GL: Fifth-Generation Lenguaje, Lenguaje de quinta generación.

Abstracción: El proceso de crear una superclase extrayendo cualidades comunes o características generales a partir de clases u objetos más específicos.

ACTOR: Un lenguaje de programación orientado a objetos con una sintaxis semejante al Pascal.

ADA: Un lenguaje de programación de alto nivel basado en Pascal y desarrollado como estándar para el departamento de defensa de los Estados Unidos de América.

Agrupamiento: El almacenamiento de objetos de forma contigüa en un disco para su acceso eficaz

Álgebra Relacional: Lenguaje de procedimientos usado para obtener nuevas relaciones de otras ya existentes por medio de operadores.

ALGOL: Lenguaje Algorítmico. Un lenguaje de programación de alto nivel, de propósito general, que se desarrolló por primera vez al principio de la década de los 60.

Análisis orientado a objetos: El análisis de las necesidades de un sistema expresado en términos de objetos del mundo real.

ANSI: American National Standards Institute. Instituto Americano de Normas o Estándares Nacionales. Una Sociedad que está dedicada al desarrollo de los estándares de la industria americana y coordina y gestiona la participación Americana en la Organización de Estándares Internacionales (International Standards Organization, ISO) y en la Comisión Electrotécnica Internacional (International Electrotechnical Commission, IEC). Los estándares son relativos a códigos de datos y comunicaciones y a los lenguajes de programación y de Administración de datos. ANSI fue fundada en 1918.

APIs: Interfaces de Programas de Aplicación.

Aplicación Cliente Servidor: Un programa o conjunto de programas cuya información procesada puede estar almacenada en varias bases de datos con diversos servidores.

Atributo: Una propiedad o característica de un objeto.

Basado en menús: Describe un programa que es gobernado seleccionando opciones a partir de una lista presentada en pantalla.

Basado en sucesos: Similar a la programación en tiempo real, un programa basado (controlado por) en sucesos tiene un bucle expedición o de envío de mensajes como nivel superior de organización. El bucle responde a todas las acciones del usuario, permitiendo a este tener un completo control de la interfaz y estando el programa subyacente preparado para responder siempre que corresponda realizar alguna acción.

Base de Datos Distribuida: Consiste en un conjunto de localidades (nodos, computadoras, etc.), cada una de las cuales mantiene un sistema de base de datos local. Las computadoras se comunican entre sí a través de diversos medios de comunicación. Además una localidad puede participar en la ejecución de transacciones globales, las localidades pueden conectarse físicamente de diversas formas. Las diferencias principales entre estas configuraciones son: costo de instalación, y el costo de comunicación. La principal ventaja de esta base de datos

es la capacidad para compartir y accesar la información de manera confiable y eficiente.

Biblioteca de clases: Es una colección de clases genéricas que pueden adaptarse y dimensionarse para una aplicación determinada.

Blob: Binary Large Object. Objeto binario de gran tamaño. Normalmente se refiere a un tipo de datos útil para almacenar grandes cantidades de datos y se utiliza frecuentemente en la comparación de relaciones y bases de datos orientadas a objetos. Un Blob se convierte en orientado a objetos solamente si los métodos son encapsulados con los datos, existe la posibilidad de manipulación de mensajes y forma parte de una jerarquía de clases.

C++: Un superconjunto orientado a objetos del lenguaje C escrito por Bajarne Stroustrup en los laboratorios Bell de A&T. El término C++, acuñado por Rick Muscatti, significa más que C.

C: Un lenguaje de programación de alto nivel desarrollado originalmente por los laboratorios Bell.

Caching: La retención de datos, en la memoria caché para lograr un rápido acceso.

CAD/CAM: Computer Aided Design/Computer Aided Manufacturing. Diseño Asistido por Computadora/Fabricación Asistida por Computadora. Productos diseñados en un sistema CAD y utilizados como entrada directa en un sistema CAM. Por ejemplo, después de diseñar una pieza en un sistema CAD (AUTOCAD), su imagen electrónica es transferida a un lenguaje de programación de control numérico (NC), el cual genera a continuación las instrucciones para controlar la pieza, Ver Robótica y MRP.

CAE: Computer Aided Engineering. Ingeniería asistida por computadora. El software que analiza automáticamente los diseños que han sido creados por los sistema CAD o introducidos de otra forma en la computadora.

Caja negra: Una metáfora de ingeniería que describe un dispositivo cuyos componentes internos son desconocidos para el usuario. Los objetos son como

cajas negras en donde sus procedimientos internos están ocultos para los usuarios y los programadores.

CAP: Computer Aided Publishing. Edición Asistida por Computadora. El empleo de paquetes de software para facilitar el diseño de texto y gráficos en aplicaciones de auto edición.

Cápsula: Un término utilizado por Metaphor Coputing Systems para describir una serie de operaciones definidas gravadas en un procedimiento. Las cápsulas pueden llamar a otras cápsulas y pueden ser públicas o privadas. Son similares al objeto en otros sistemas pero sin herencia.

CASE: (Computer Aided Software Engineering), Ingeniería del Software (o de sistemas) Asistida por Computadora. El empleo de paquetes de software que ayudan en una o más fases del ciclo de vida del sistema (Análisis, Diseño, Construcción o mantenimiento).

CD-ROM: Compact Disk Read Only Memory. Disco Compacto-Memoria de sólo lectura. Un disco compacto con un formato especial utilizado para guardar datos del tipo multimedia, que incluyen Texto Gráficos y Video en movimiento, así como audio. Los discos CD-ROM guardan más de 600 megabytes de datos.

CD: Compact Disk. Disco Compacto. Disco que contiene hasta 72 minutos de sonido estéreo de alta fidelidad. Es un dispositivo de acceso directo, y las selecciones individuales pueden ser reproducidas en cualquier secuencia.

CIM: Computer Integrated Manufacturing. Fabricación Integrada por computadora. La integración de la fabricación (factoría) automatizada (como por ejemplo la programación de máquinas y la fabricación de sistemas de contabilidad, como ejemplo, el costo de los materiales.

Clase derivada: Una subclase en C + +, ver subclase.

Clase hija: Ver subclase.

Clase Padre: Ver superclase.

Clase: La descripción de un conjunto de objetos casi idénticos que comparten métodos comunes y características generales.

Cliente/Servidor: Una relación entre máquinas de una red de comunicaciones. El cliente es la máquina solicitante; el servidor es la máquina suministradora.

CLOS: Common Lisp Object System. Sistema de Objetos Comunes de Lisp. Una versión orientada a objetos del lenguaje de programación de inteligencia artificial LISP.

Datos: La palabra datos (del latín data, plural de datum) significa simplemente "hechos", es decir entidades independientes sin evaluar. Los datos pueden ser numéricos o no numéricos (por ejemplo alfabéticos o simbólicos).

DB2: Un DBMS Relacional de IBM para equipos minis.

DBMS: (Database MAnegement System), es un conjunto de programas que controlan la organización, el almacenamiento y la recuperación, la seguridad y la integridad de una base de datos. Normalmente existen tres componentes en un DBMS: Un Lenguaje de definición de datos que describe los datos, un lenguaje de manipulación de datos que describe lo que el usuario desea hacer con los datos (por ejemplo crear, actualizar, borrar, etc.) y un lenguaje de control de datos que describe quién puede acceder y cómo y cuando se produce el acceso.

DDL: (Data Definición Language) Lenguaje de definición de datos.

Declaración: Concepto referido al almacenamiento y ejecución de procedimientos como parte de una base de datos orientada a objetos. Una declaración es una propiedad del objeto con la que típicamente se asocia alguna condición que el objeto debe satisfacer. La declaración es parecido a un método pero no está encapsulado junto con los datos locales sobre los que opera.

Demonio: Un objeto activo.

Depurador: Un programa que permite a los usuarios corregir los errores del programa examinando y modificando el contendido de la memoria e iniciando o deteniendo la ejecución en un punto predeterminado o "punto de ruptura".

Depurar: Localizar y reparar errores en la lógica de un programa.

Diccionario de datos: Documentación de la base de datos, creada como una base de datos por sí misma y es llamada algunas veces metadatos.

Diseño orientado a objetos: La traducción de la estructura lógica de un sistema a una estructura física compuesta de objetos de software.

Disparadores: Concepto referido al almacenamiento y ejecución de procedimientos como parte de una base de datos orientada a objetos. Un disparador es un procedimiento que se activa automáticamente siempre que surja una condición predeterminada. El disparador es parecido a un método pero no está encapsulado junto con los datos locales sobre los que opera.

DLL: Dinamic Link Libraries. Bibliotecas de enlace dinámico. Ver ligadura dinámica.

ECAD: Electronic Computer Aided Design. Diseño Eléctrico Asistido por Computadora, es decir, CAD para diseño electrónico.

Eiffel: Un Lenguaje orientado a objetos Derivado de Smalltalk y desarrollado por Bertrand Meyer.

Embebido: Embedded. Inmerso, empotrado.

Encapsulación: El conjunto de métodos y variables modelo dentro de una clase u objeto, de forma que el acceso a las variables está permitido solamente a través de los métodos propios del objeto.

Enlazador (linker): Un paso de la compilación que consiste en unir dos o más módulos.

Entorno ventana: Una computadora que está operando con un ambiente gráfico o un sistema operativo que proporciona múltiples ventanas en pantalla DESQview, Windows de Microsoft, Presentation Manager, Finder, Multifinder, y X-Windows son ejemplos de entornos de ventanas.

Esquema: Define un modelo de base de datos conceptual por medio de la definición no sólo de los campos y los registros, sino también de las relaciones entre los datos dentro de los diferentes registros.

Extensibilidad: La capacidad de un programa o sistema para ser fácilmente alterado de forma que pueda tratar con nuevas clases de entrada.

Flavors: Una ampliación orientada a objetos de LISP perteneciente a Graphael.

G-Base: Una base de datos orientada a objetos basada en LISP, perteneciente a Graphael.

GDI: Interfaz de Dispositivos Gráficos.

GUI: Interfaz Gráfica del usuario, es un conjunto de programas que entre sus funciones pueden listarse: la captura y presentación de la información en la pantalla con el formato indicado, el manejo de ventanas, el control del teclado y del Mouse, etc.

Gemsone: Una base de datos orientada a objetos perteneciente a Servio Logic, Inc.

Generación de esquema: La generación automática de un esquema de base de datos, o la descripción de sus registros y campos y de la forma en que están relacionados, a partir de un lenguaje de definición de datos "DDL".

Gestión de memoria: La forma en que una computadora trata su propia memoria. Incluye protección de memoria y cualquier memoria virtual o técnicas de intercambio de memoria.

Granulación: El nivel de modularidad de un sistema. Una modularidad mejor indica módulos más pequeños y proporciona una flexibilidad mayor.

Hashing (distribución segmentada): Un método o algoritmo diseñado especialmente para convertir claves en direcciones. El método más sencillo es el de la división en este método, la clave se divide entre un número cercano al número de registros en el archivo, y el residuo de la división se toma como la dirección relativa.

Herencia: Un mecanismo para compartir automáticamente métodos y tipos de datos entre clases, subclases y objetos. No se encuentra en los sistemas procedimentales o por procedimientos. La herencia permite programar sólo la gestalt (lo que es diferente) de las clases definidas previamente.

Herencia múltiple: La capacidad de las subclases para heredar variables modelo y métodos desde más de una clase. Es muy útil para construir un comportamiento compuesto desde más de una rama de una jerarquía de clases.

I-CASE: (Integrated Computer Aided Software Ingenering), Ingeniería del Software Integrada Asistida por Computadora.

IDE: Ambiente Integrado de Desarrollo.

Identidad de un objeto: Es el identificador único de un objeto lo que permanece invariable en un objeto a lo largo de todas las posibles modificaciones de su estado. Puede utilizarse para marcar el objeto.

Información: Es un conjunto ordenado de datos, los cuales pueden recuperarse de acuerdo con la necesidad del usuario.

Ingeniería del software: Si Software es "programas de computadora, procedimientos y documentación y datos pertinentes a la operación de un sistema de computo" e ingeniería "la aplicación sistemática del conocimiento científico a la creación y construcción de soluciones económicamente rentables a problemas prácticos puestas al servicio de la especie humana", entonces Ingeniería del software es "La rama de la ingeniería que aplica los principios de las ciencias de la computación y las matemáticas para lograr soluciones rentables a los problemas del desarrollo del software"

Instancia, modelo, caso: Un objeto que es miembro de una clase.

Integridad de datos: El estado de los datos que han sido protegidos contra un borrado accidental o una modificación in controlada.

Interfaz orientado a objetos: Una interfaz basada en la manipulación directa de objetos tales como íconos y menús.

interfaz por íconos: Interfaz de usuario que visualiza objetos en pantalla como imágenes diminutas, las cuales se pueden señalar y seleccionar por medio de un "ratón".

Jerárquico: Descripción de un sistema que tiene una estructura constituida por diferentes niveles, en la que los niveles superiores controlan o tienen prioridad sobre los niveles inferiores.

Lenguaje concurrente: Un lenguaje que admite o soporta la ejecución simultánea de múltiples objetos, normalmente en equipos de arquitectura paralela.

Lenguaje de alto nivel: Lenguajes de programación independientes de la máquina, opuestos a los lenguajes ensamblador y de máquina que se consideran de bajo nivel.

Lenguaje no procedimental (no proceduall): Un lenguaje de programación que genera directamente la lógica (código) del programa a partir de la descripción del problema que hace el usuario, en lugar de generarla a partir de un conjunto de procedimientos basados en la lógica tradicional de programación.

Lenguaje Orientado a objetos: Un lenguaje de computadora que soporta objetos, clases, subclases, mensajes y herencia. Las características secundarias pueden incluir herencia múltiple, ligadura dinámica y polimorfismo.

Lenguaje procedimental (proceduall): Un lenguaje de programación como COBOL, FORTRAN, BASIC, C, PASCAL, basados en el empleo de un orden apropiado de acciones y en el conocimiento de las operaciones de proceso de datos y de las técnicas de programación.

Ligadura dinámica: La realización o puesta en práctica, consecuencia de la herencia y el polimorfismo. Cuando se ejecuta un programa orientado a objetos, los mensajes los reciben los objetos. El método para tratar un mensaje se almacena con frecuencia en una biblioteca de clases. El método se localiza dinámicamente cuando se necesita y la ligadura se produce entonces en el último instante posible. Ver ligadura.

Ligadura estática: Ligadura temprana, ver Ligadura.

Ligadura tardía: Ligadura dinámica, ver Ligadura.

Ligadura temprana: Ligadura estática, ver Ligadura.

Ligadura: El proceso de entrelazar un programa para resolver todas las conexiones entre sus componentes. La ligadura estática o temprana, resuelve estas conexiones antes de ejecutar el programa. La ligadura dinámica o tardía, sucede cuando se está ejecutando el programa.

LISP: List Processing Languaje. Lenguaje de procesamiento de listas. Un lenguaje de programación de segunda generación de inteligencia artificial que contribuyo a la ligadura dinámica y un entorno de desarrollo interactivo para la evolución de lenguajes de programación orientados a objetos.

Llamada (Call): Una sentencia en un programa que referencia una subrutina o programa independiente. Después de terminar su proceso, la rutina o programas llamados vuelven al programa o rutina que les invocó.

Llave Primaria: Campo cuyo contenido puede identificar de manera única cada registro del archivo.

Macintosh: Una línea de computadoras personales de Apple Computer basadas en microprocesadores 6800 de 32 bits de Motorola. Lanzado inicialmente en 1984, el Macintosh ejecuta un sistema operativo propietario que incluye un escritorio (desktop) simulado. El Macintosh soporta gráficos de media y alta resolución y dispone de un lenguaje incorporado para gráficos.

Mapeo: Transformación de datos de una forma y un contexto, en otra forma y otro contexto. Por ejemplo, registros lógicos se mapean en registros físicos para su almacenamiento.

Marco estructural (framework): Una biblioteca de clases que está preparada especialmente para una categoría determinada de aplicación.

MCAD: Mechanical Computer Aided Dessign. Diseño mecánico asistido por computadora. es decir CAD para diseño mecánico.

Memoria caché: Del francés Cahe (Bolsa de mano) Es una sección de memoria donde se almacena lo que se emplea con más frecuencia. (1) Una sección reservada de la memoria principal (RAM). (2) Un banco independiente de la memoria de alta velocidad que actúa como una memoria intermedia entre la memoria principal y la Unidad Central de Proceso para mejorar el rendimiento.

Mensaje: Una solicitud enviada a un objeto para cambiar su estado o entregar un valor. El mismo mensaje puede enviarse a objetos diferentes porque los mensajes indican simplemente a un objeto lo que tiene qué hacer. Los métodos, definidos dentro del objeto receptor, determinan cómo efectuará el objeto la solicitud o petición. Ver Método y Polimorfismo.

Método: La función o procedimiento que pone en practica o realiza la respuesta cuando se envía un mensaje a un objeto. Los métodos determinan la forma en que un objeto responderá a un mensaje que recibe. Ver Mensaje.

MIPS: Millions of Instructions Per Second. Millones de instrucciones por segundo. Una medida de la potencia de cálculo o computación. Un computador de gran tamaño puede realizar normalmente de 10 a 50 MIPS; un microprocesador podría estar situado en el intervalo de 0.05 MIPS.

Modelo de Entidad -Relación: Método empleado en el diseño de bases de datos basado en el análisis de tres modelos semánticos claves: entidades, relaciones y atributos.

Modificación de esquema: El proceso de reestructurar una base de datos.

MRP: Planeamiento de Resistencia de Materiales.

Normalizar: La descomposición de datos en grupos de registros para lograr un proceso eficaz en un sistema de base de datos relacional.

Núcleo (Kernel): Una parte fundamental de un programa, como por ejemplo un sistema operativo, que reside en memoria en todo momento.

Objective-C: Un lenguaje de programación orientado a objetos perteneciente a The Stepstone Corporation, formado añadiendo construcciones inspiradas en Smalltalk al lenguaje C.

Objeto activo (active object): Un objeto que realiza el seguimiento de los sucesos que ocurren en una aplicación y actúa de forma autónoma. A veces se denomina agente o demonio.

Objeto pasivo: Un objeto que actúa solamente a petición, como por ejemplo los botones de windows que deben pulsarse para realizar una acción.

Objeto: El elemento primitivo en la programación orientada a objetos. Los objetos son entidades que encierran dentro de ellas tanto los datos que describen el objeto como las instrucciones para operar con dichos datos.

Ocultación de información: Una estrategia de diseño cuyo objetivo es maximizar la modularidad ocultando tanta información como sea posible dentro de los componentes de un diseño.

ODBC: Open Data Base Conectivity. Es un driver estándar que permite conectar bases de datos generalmente de Windows con sistemas de bases de datos más grandes en Servidores.

OPAL: Operational Performance Analysis Languaje. Lenguaje de análisis del rendimiento operativo. Es un DDL/DML lenguaje de definición de datos/lenguaje de manipulación de datos exclusivos de GemStone de Servio-Logic.

Orientación del objeto: Un nivel de estrategia de diseño, cuyo objetivo es maximizar la modularidad ocultando tanta información como sea posible dentro de los componentes de un diseño.

Padre-Hija: Una forma de expresar la relación entre clases y subclases. Las clases o subclases hijas heredan los métodos y variables modelo de la clase padre. Con herencia múltiple, una clase hija puede tener varios padres.

Paradigma: Un paradigma en el sentido de los paradigmas científicos es una manera de visualizar la realidad que sirve como base para toda una corriente de pensamiento científico. Respecto a la computación un paradigma entonces, es un modelo mental que involucra una conceptualización especial de qué es un problema y cómo representar el proceso de su solución por medio de la computadora

Paso de mensajes: Un mecanismo que permite a los objetos enviarse mensajes entre ellos.

Persistencia: La permanencia de un objeto, especialmente importante en el contexto de bases de datos orientadas a objetos, que mantiene una distinción entre los objetos creados solamente para la duración de la ejecución y aquellos que están pensados para un almacenamiento permanente.

Polimorfismo: La capacidad del mismo mensaje para ser interpretado de forma diferente al ser recibido por objetos diferentes. El mensaje "Print", por ejemplo, cuando se envía a una figura o diagrama, invoca un método o realización diferente

del que se invoca cuando se envía el mismo mensaje "Print" a un documento de texto. Ver Mensaje.

Preprocesador: Un programa que realiza cierto proceso preliminar sobre la entrada de datos antes de que sea procesada por el programa principal. Por ejemplo, el código fuente de C++ se preprocesa normalmente o se traduce a código fuente de C antes de compilarse.

Procesamiento distribuido: Un sistema de computadoras conectadas entre sí mediante una red de comunicaciones en la que cada una de las computadoras administra su propia carga de trabajo y la red apoya al sistema como un todo.

Programas: Un programa no es más que una serie de instrucciones que le dicen a la computadora que efectúe acciones específicas.

Programación descendente (top-down): Una metodología que produce un programa modular, estructurado jerárquicamente. El diseñador crea en primer lugar los códigos y comprueba un módulo de nivel superior que representa la estructura general del programa y continúa después de la misma forma para crear módulos de nivel inferior que representan sus subfunciones. Ver Programación estructurada.

Programación estructurada: Una filosofía de programación enfocada a administrar la complejidad mediante el establecimiento y normalización de una metodología de programación. La programación estructurada es un método descendente (top-down): El programa se descompone en su forma más simple y general; se divide en módulos independientes y más pequeños; y los componentes del programa se organizan en una estructura jerárquica.

Programación orientada a objetos: Una metodología para crear programas utilizando conjuntos de objetos auto-suficientes que tienen datos y comportamiento encapsulados, actuando a petición, e interactuando con cada uno de ellos mediante el paso de mensajes en ambos sentidos.

Programación tradicional: La programación que utiliza lenguajes procedimentales como FORTRAN, COBOL y BASIC. Dichos lenguajes admiten una construcción del programa basada en la determinación de la secuencia de los procedimientos que

actúan sobre un conjunto independiente de datos. La programación tradicional se realiza en tres fases secuenciales: diseño, realización o práctica (implementación) y comprobación o prueba.

Programación visual: Una categoría general de aplicaciones que hacen gráfica la programación y sus efectos visibles al usuario. Por ejemplo en los programas de dibujo, los objetos pueden dibujarse, comprimirse y modificarse manipulando directamente la imagen en pantalla, en lugar de cambiar datos numéricos en una tabla de dimensiones.

Ranuras: Las propiedades o atributos de un objeto.

RDBMS: Relational Databse Managament System. Sistema Manejador de Bases de Datos Relacinales.

Recolección de basura: Una rutina de Administración de memoria que busca en la memoria segmentos de programas, datos u objetos que no estarán activos en el futuro y recuperá el espacio sin utilizar.

Recursividad: La capacidad de una rutina o programa de llamarse a sí misma.

Servidor de Archivos: En una red de área local, una computadora que almacena los programas y los archivos de datos compartidos por los usuarios conectados a la red.

Servidor de base de datos: Una computadora independiente en una red de área local que almacena y administra la base de datos. Operaciones tales como la recuperación de registros se realizan en el server. Es totalmente diferente a un servidor de archivos, que actúa como una unidad de disco a distancia (remota) y que requiere que se transmitan partes importantes de la base de datos a la computadora del usuario para dichas operaciones.

Simula: Un lenguaje de programación desarrollado a finales de los años sesenta para simulación de programación. Simula incluía mecanismos de clase y herencia, se considera que Simula ha influenciado en gran manera a los lenguajes de programación orientados a objetos.

Smalltalk/ V: Una versión del lenguaje Smalltalk de Digitalk Corporation.

Smalltalk: El primer lenguaje y entorno de programación verdaderamente orientado a objetos. Desarrollado en Xerox PARC a finales de los años setenta, fue utilizado originalmente para crear prototipos de lenguajes de programación más sencillos e interfaces gráficos basados en ventanas. Su entorno integrado elimina la distinción entre el lenguaje de programación y el sistema operativo y permite al programador personalizar la interfaz con el usuario y el comportamiento del sistema.

SQL: Structured Query Language. Lenguaje de Consulta Estructurado. Un lenguaje diseñado para consultar datos en una base de datos relacional.

Subclase: El refinamiento de una clase en otra más especializada. A veces se denomina clase derivada o hija. Los métodos o tipos de datos comunes se almacenan en una clase tan abstracta como sea posible de forma que puedan ser heredados por todas las subclases relevantes.

Superclase: En una jerarquía de herencia, una clase más general que almacena variables y métodos que pueden ser heredados por otras clases. A veces se le denomina clase base o padre.

Tipeado estático: La adición de tipos a cada objeto en la compilación. con el tipeado (construcción de tipos) estático, la búsqueda en ejecución del método apropiado a un mensaje se restringe a la clase y superclase del receptor, asegurando de esta forma que siempre se encontrará un método apropiado.

Transparente: Algo transparente parece no existir pero existe.

Two-phase commit (confirmación de dos fases): Una característica de los DBMS Distribuidos que permiten asegurar la integridad de las transacciones actualizadas en ambientes distribuidos.

Tupla (tuple): Un registro, o fila, en el contexto de bases de datos relacionales.

Unión: La comparación de un archivo de una base de datos (tabla) frente a otro, basada en alguna condición, y la creación de un tercer archivo conteniendo los datos de los archivos comparados. Por ejemplo un archivo de vendedores puede unirse con uno de ordenes de compra para crear un archivo de todos los vendedores de un producto.

UNIX: Un sistema operativo multitarea y multiusuario de A&T que se ejecuta en una amplia variedad de sistemas de computadoras.

Variable de instancia: Los datos contenidos dentro de un objeto que describen las propiedades que posee el objeto.

Variable global: Una variable accesible por todos los módulos de un programa.

Variable: Una estructura de memoria que guarda los datos que le han sido asignados hasta que recibe un nuevo valor o termina el programa.

Variable de instancia: Sinónimo de campo, slots, atributos miembros de datos.

Vbase: Una base de datos orientada a objetos de Servio-Logic.

Virtual: Un entorno simulado o conceptual. por ejemplo, la realidad virtual es arealidad simulada. Parece existir pero en realidad no existe.

BIBLIOGRAFIA

BIBLIOGRAFIA

- ANALISIS Y DISEÑO ORIENTADO A OBJETOS.
  James Martin, James J. Odell.
  Edit. Prentice Hall, México, 1994; ISBN 0-13-630245-9.

- BASES DE DATOS ORIENTADAS A OBJETOS (APUNTES DE SIMPOSIUM).
  MARCOS CALDERON MACIAS.
  FUNDACION ARTURO ROSENBLUETH.
  NOVIEMBRE DE 1994.

- TECNOLOGIA PARA EL CAMBIO: LA INGENIERIA DEL SOFTWARE CON LA ORIENTACION A OBJETOS (APUNTES CURSO INTENSIVO).
  FUNDACION ARTURO ROSENBLUETH.
  SEPTIEMBRE DE 1994.

- BASES DE DATOS ORIENTADAS A OBJETOS (APUNTES MAESTRIA).
  OFELIA CERVANTES VILLAGOMEZ.
  UNIVERSIDAD DE LAS AMERICAS, PUEBLA.
  VERANO DE 1993.

- ANALISIS Y DISEÑO CON LA METODOLOGIA DE OBJETOS.
  HANNA OKTABA.
  APUNTES DE LA MAESTRIA EN CIENCIAS DE LA COMPUTACION. UNIVERSIDAD NACIONAL AUTONAMA DE MEXICO.
  VERANO DE 1993.

- MODELACION DEL MUNDO CON OBJETOS (APUNTES CONFERENCIA).
  MANUEL COTA AGUILAR.
  ONTICA S.A. DE C.V. Y FUNDACION ARTURO ROSENBLUETH.
  NOVIEMBRE DE 1994.

- OBJECT ORIENTED DATABASE MANAGEMENT.
  APPLICATIONS IN ENGINEERING AND COMPUTER SCIENCE.
  Alfons Kemper, Guido Moerkotte.
  Edit. Prentice Hall, New Jersey, 1994 ISBN 0-13-629239-9.

- SOFTWARE ORIENTADO A OBJETOS.
  Ann L. Winblad, D, Samuel D. Edwards, David R. King.
  Edit. Addison-Wesley/Diaz de Santos, U.S.A, 1993; ISBN 0-201-60117-6.

- VERSANT OODBMS SYSTEM REFERENCE MANUAL.
  RELEASE 2.
  VERSANT OBJECT TECHNOLOGY CORPORATION MAY, 1993.

- OBJECT-ORIENT INFORMATION SYSTEMS.
  PLANNING AND IMPLEMENTATION.
  David A. Taylor, PhD.
  Edit. Wiley Professional Computing, 1992; ISBN -471-54364-0.

- OBJECT ORIENTED DATABASES:
  TECHNOLOGY, APPLICATIONS, AND PRODUCTS.
  BINDU R. RAO.
  Edit. Mc. Graw-Hill, 1994.

- INGENIERA DEL SOFTWARE.
  Roger S. Pressman.
  Edit. Mc. Graw-Hill, México, 1993.

- TECNICAS DE BASES DE DATOS.
  (ESTRUCTURACION EN DISEÑO Y ADMINISTRACION).
  Atre Shakuntala.
  Edit. Trillas, México, 1988.

- SISTEMAS DE BASES DE DATOS.
  (ADMINISTRACION Y USO).
  Alice Y. H. Tsai.
  Edit. Prentice Hall, México, 1990.

- INTRODUCCION A LOS SISTEMAS DE BASES DE DATOS.
  C.J. Date.
  Edit. Adison Wesley Iberoamericana, 1993.

- ANALISIS DEL AREA EMPRESARIAL, TECNICAS EN EL ANALISIS DE FUNCIONES
  JAMES MARTIN & Co. 1992.

- COMPUTER WORLD MARZO 27-31, 1995 (PERIODICO DE COMPUTACION).

- SOLUCIONES AVANZADAS, TECNOLOGIAS DE INFORMACION Y ESTRA-TEGIAS DE NEGOCIOS. AÑO 1, Nº 6 NOVIEMBRE-DICIEMBRE 1993, Nº 15 NOVIEMBRE DE 1994. AÑO 3, Nº 22 1995, (REVISTA DE COMPUTACION).

- DATA BASED ADVISOR JANUARY, FEBRUARY 1995. (REVISTA DE COMPUTACION).

- PERSONAL COMPUTING MEXICO AGOSTO 1993 (REVISTA DE COMPUTACION).

- OBJECT MAGAZINE OCTOBER 1994. (REVISTA DE COMPUTACION).

- OBJECTS IN EUROPE VOLUME 1, NUMBER 4, AUTUMN 1994.

- JOURNAL OF OBJECT ORIENTED PROGRAMMING, VOLUMEN 6, 25-28-1993.