UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

Aragón

ANALISIS COMPARATIVO ENTRE DOS SISTEMAS DE ADMINISTRACION DE BASES DE DATOS

T E S I S

Que para obtener el Título de:

INGENIERO EN COMPUTACION

Presenta:

ISRAEL LAUREANO LAZCANO

Director de tesis: Ing. Victor Hugo Salazar Nogales

San Juan de Aragón, Estado de México 1992

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE DE MATERIAS

LISTA DE FIGURAS.

PREFACIO

En los años recientes, se ha generado un gran entusiasmo e interés por una nueva metodología de la ingeniería de software llamada programación orientada a objetos (POO o por sus siglas en inglés, OOP). Todas las revistas dedicadas al tema de la computación han publicado muchos artículos relacionados a la programación orientada a objetos; hasta han surgido revistas totalmente dedicadas a esta metodología; además, se han hecho muchos seminarios, conferencias, simposiums y exposiciones dedicados a la orientación a objetos, y se siguen realizando con una frecuencia cada vez mayor. Este trabajo es una muestra del interés que despierta este tema.

Cabe hacer la siguiente aclaración: la frase "programación orientada a objetos" está mal aplicada, se debería hablar de "diseño orientado a objetos", aunque muchos utilizan ambas frases para nombrar diferentes etapas de desarrollo. Este tema está tratado con mayor amplitud a lo largo de esta tesis.

A través de este libro también hay desarrollos y aclaraciones acerca de la importancia de esta nueva metodología y del nuevo paradigma que propone. En forma resumida, la popularidad e importancia de esta metodología radica en la sencilla observación de que los programas de computadora realizan acciones en objetos. Así de simple. Dejando a un lado la definición de lo que es un "objeto", debemos notar que casi todo el software convencional es solamente una lista de acciones que se realizarán en ciertos datos en una cierta secuencia; si llega a cambiar alguna acción o el tipo de dato que maneja esa acción, se estará cambiando todo el programa. La metodología orientada a objetos (para evitar polémicas acerca del uso de "programación orientada a objetos" o "diseño orientado a objetos" o "OOP" o "OOD") ofrece la oportunidad para llevar a cabo estos cambios, datos o acciones o ambos, extendiendo un programa o agregándole cosas, en vez de cambiarlo.

Uno de los campos de aplicación de la programación de computadoras más importantes desde el inicio de la era informática es el almacenamiento y administración de grandes volúmenes de datos, es decir, el uso de administradores de bases de datos. El enfoque que se ha seguido hasta ahora en cuanto a bases de datos es el descrito en el párrafo anterior: acciones que se realizan sobre ciertos datos. El enfoque orientado a objetos aplicado a la base de datos es el tema que se explora a lo largo de este trabajo; lo que ha resultado el encuentro de la metodología orientada a objetos y la administración de bases de datos.

La tesis está dividida en cinco capítulos:

Capítulo 1. Objetivos y Propósito de la Tesis. Acerca de los objetivos que se proponen en este trabajo, la estrategia de trabajo que se seguirá y las causas e importancia que tiene o puede tener este trabajo.

Capítulo 2. Introducción a la Programación Orientada a Objetos. En este capítulo se desarrolla un marco teórico con las definiciones y técnicas de la metodología orientada a objetos, además de proporcionar una base para el desarrollo del análisis y diseño del sistema.

Capítulo 3. Teoría de Bases de Datos. Este capítulo está enfocado a la definición de la terminología de bases de datos y el desarrollo que hay en este aspecto. Se enfatizan, sobre todo, los temas relacionados al modelo relacional para bases de datos y los modelos alternativos que pueden sustentar a las bases de datos según la metodología orientada a objetos.

Capítulo 4. Análisis Comparativo y Diseño de un Sistema de Administración de Bases de Datos Orientado a Objetos. Este tema es concretamente el proyecto propuesto en esta tesis. Basado y desarrollado según los conceptos explicados en los capítulos anteriores.

Capítulo 5. Conclusiones y Trabajos Posteriores. En este capítulo se presentan las conclusiones a las que se llegan según los objetivos marcados al principio de la tesis y al análisis y diseño desarrollados a lo largo del trabajo. Trabajos posteriores se refiere al análisis de los trabajos que podrían dar continuidad a este tema enfocándose a algún campo de desarrollo.

CAPITULO 1

Objetivos y Propósito de la Tesis.

1.1. TITULO Y JUSTIFICACION.

Esta tesis lleva como titulo "Análisis Comparativo entre dos Sistemas de Administración de Bases de Datos"; se trata de la comparación entre un sistema informático para el manejo de bases de datos que utiliza conceptos convencionales para su manejo e implantación y un sistema de administración de bases de datos implantada a través de técnicas de programación orientadas a objetos, y proponiendo nuevas formas de administración de datos.

Una de las aplicaciones principales de la computación es la creación y manejo de bases de datos, es decir, el manejo de archivos de datos de forma que se presente información que pueda ser útil en un momento dado. Estos sistemas pueden seguir una filosofía que se enfoque a los procesos a seguir para el funcionamiento del sistema, y también se puede seguir una filosofía que esté enfocada en los elementos o entidades que conforman el sistema. Este trabajo trata de la comparación entre ambas filosofías o maneras de funcionamiento del sistema.

1.2. DELIMITACION DEL CAMPO DE DESARROLLO.

La ingeniería en computación abarca, básicamente, dos áreas de conocimiento: electrónica e informática, llamadas también hardware y software. Estos campos de la computación se complementan entre sí y a veces llega a confundirse la frontera que los separa

Dentro del área de la ingeniería informática o ingeniería de software existen diversas aplicaciones y disciplinas; puede hablarse de los conceptos o reglas que se definen la ingeniería de software en forma abstracta o general, por ejemplo el concepto de modularidad o el ocultamiento de información. También existe el tema de las diferentes técnicas de programación; es decir, detallar o explicar como se podría implantar alguna operación en el dominio de la información. Dentro del campo de la informática también se halla la disciplina del estudio de la aplicación de diversas teorías de la información al mundo de la computación: programación usando goto's, estructuración, orientación a datos, orientación a objetos, etc.

Dentro del campo de las aplicaciones informáticas, se pueden hallar sistemas de hoja de cálculo, sistemas de dibujo, sistemas de procesamiento de palabras, sistemas de auto-edición, edición de escritorio o de publicación personal. Una de las aplicaciones más importantes es el manejo o administración de bases de datos, y ese tipo de aplicación es el tema principal sobre el que se apoya este trabajo.

1.3. OBJETIVOS.

El objetivo principal de la tesis es el mostrar el diseño o la forma de funcionamiento de un administrador de bases de datos. Este diseño de administradores de bases de datos se hace siguiendo dos paradigmas de funcionamiento: un paradigma o modelo de funcionamiento está enfocado a los procedimientos que procesan los datos necesarios. Es el llamado paradigma orientado a procedimientos; el otro paradigma está enfocado a los elementos que van a conformar el sistema, datos y

procedimientos, es el llamado paradigma orientado a objetos.

El analisis comparativo se hace con los dos diseños del mismo sistema de administracion de bases de datos, usando los dos paradigmas de funcionamiento.

1.4. HIPOTESIS.

1.4.1. Importancia.

El desarrollo de sistemas de programación , en la actualidad, se presenta mucho más complejo, variado y extenso. Los sistemas deben tener una interfaz gráfica para comunicación con el usuario, y deben poder manejarse con mucha facilidad, por lo menos desde el punto de vista del usuario; desde el punto de vista de su diseño y codificación, los sistemas deben volverse "a prueba de balas" (esto es, deben poder recuperarse de cualquier error que pudiera presentarse) y ser de una complejidad impresionante, puesto que el código desarrollado es extenso y modular (es decir, el sistema se desarrolla principalmente en forma de intercomunicación entre módulos ejecutables); el diseño debe especificar un sistema que pueda cubrir con éxito una amplia gama de requerimientos.

En el aspecto del desarrollo de sistemas, se pide mayor interactividad de los mismos, mayor capacidad de respuesta, mayor tolerancia hacia los (posibles) errores de los usuarios, manejo de simbología gráfica, tanto para la comunicacion máquina-usuario como para el manejo de la información. En el aspecto económico de los sistemas de información, se piden costos de desarrollo mucho mas reducidos, un tiempo de entrega mucho menor, los sistemas deben cumplir todos los requerimientos pedidos y deben cumplirlos bien, sin que el sistema presente defectos en el diseño en cuanto a su funcionamiento, además, la competencia entre las diferentes compañías de desarrollo es muy enconada y los sistemas, aparte de todo lo mencionado, deben tener un muy alto grado de confiabilidad, poder y facilidad de uso.

Uno de los temas que se tocarán durante el desarrollo de esta tesis, habla acerca de que las bases de datos orientadas a objetos proponen la formación de objetos comunes a un sistema; esto es, los datos que se manejen a través del sistema manejador de bases de datos, pueden ser los mismos datos que se manejen a través de una hoja de cálculo, o un procesador de textos, o un sistema de agenda electrónica. En un sentido amplio, se habla de que la teoría orientada a objetos facilita el camino para la formacion de software estandar, es decir, que los programas se puedan intercomunicar o compartir la misma información.

Las técnicas de diseño y programación orientados a objetos (conocidos como OOD y OOP) proporcionan herramientas para poder afrontar esta clase de nuevos sistemas.

En este punto precisamente radica la importancia fundamental del tema propuesto en esta tesis: el mostrar la viabilidad práctica del uso de estas técnicas. La ejemplificación de estas técnicas se hace

comparativamente, es decir, los conceptos principales se muestran a través de diferencias entre las características que pudiera poseer una aplicación desarrollada a través de técnicas orientadas a procedimientos y técnicas orientadas a objetos.

1.4.2. Alcance y Significado.

La hipótesis sustentada es el análisis y diseño de un administrador de base de datos usando las técnicas definidas por el paradigma orientado a objetos.

Esta hipótesis significa que el diseño que se hace siguiendo los lineamientos marcados por el paradigma orientado a procedimientos puede ser diferente al diseño hecho usando las técnicas orientadas a objetos.

1.4.3. Enfoque Usado.

El tema de tesis está enfocado principalmente a hacer un análisis teórico-práctico de las diferencias entre el diseño y manejo de un sistema de administración de bases de datos tradicional y un sistema diseñado a través de técnicas orientadas a objetos. Aunque se trata el tema del diseño y programación orientadas a objetos, y se mencionan los diferentes modelos de desarrollo (tradicionales) de administradores de bases de datos, el enfoque usado es práctico.

A lo largo de este trabajo se desarrollan temas procurando usar ejemplificaciones y analogías apropiadas para los paradigmas de programación y las bases fundamentales en las que se apoyan los diferentes modelos de bases de datos.

CAPITULO 2

Introduccion a la Programacion Orientada a Objetos.

2.1. TEORIA DE LA ORIENTACION A OBJETOS.

2.1.1. ANTECEDENTES.

Hace más de 20 años, en 1968, durante las Conferencias sobre Ingeniería de Software de la OTAN, se acuñó el término crisis del software para indicar el hecho de que el software era, en ese momento, escaso, caro, de insuficiente calidad, era difícil mantenerlo dentro de un calendario fijo, e imposible de manejar [COX90].

La estructuración de los modos de programación, junto a la disciplina de trabajo que implica el paradigma estructuralista aliviaron la entonces enorme crisis del software. La revolución estructurada cambió el modo de programar, además de todas las actividades que implica el desarrollo de un proyecto de sistema: el diseño de algoritmos y de la filosofía de trabajo del sistema, la administración del proyecto, etc. En general, y lo más importante, cambió la forma en que se entendía el software, cambió la manera de pensar de todos los programadores; en términos técnicos, se dice que cambió el paradigma de la programación.

En las últimas décadas se ha detectado un incremento en las expectativas que los usuarios tienen hacia el software. Aparte de eso, los sistemas informáticos empezaron a crecer y a competir entre sí en cuanto a las características que ofrecían. La falta de capacidad para manejar sistemas complejos generalmente se refleja en proyectos retrasados, totalmente fuera de presupuesto, y deficientes en el cumplimiento de los requerimientos pedidos. Es obvia la analogía entre esta crisis de software y la que existió a finales de la década de los 60's.

El problema actual involucra no sólo el alto costo de desarrollo de software, sino también la calidad tan pobre del software existente; lo anterior se manifiesta en el hecho de que muchas organizaciones gastan del 40% al 75% de sus recursos en el mantenimiento de software viejo, lo que incluye la eliminación de errores lógicos (bugs) y la actualización de requerimientos cambiantes. (Ver figura 2.1).

¿Cómo podemos reconciliar la alta demanda por software de calidad con la realidad de los escasos recursos para el desarrollo? La solución puede estar en las habilidades que empleamos para manejar la complejidad en la vida diaria; formalmente: descomposición, abstracción, construcción de herramientas y composición. Estas habilidades también pueden ayudar a cumplir con los retos del desarrollo del software [BOOB7].

En tiempos recientes se ha propuesto una nueva técnica de programación, que probablemente cumpla con todas las características para resolver (o por lo menos paliar) la crisis de software: la orientación a objetos. (En realidad, las técnicas orientadas a objetos han existido desde hace tiempo, esto se explicará en la historia de la programación orientada a objetos). La orientación a objetos posee mejores métodos y herramientas para:

- Modelar el mundo real tan cercanamente como es posible a la perspectiva de un usuario.
- Interactuar fácilmente con un medio ambiente computacional, usando metáforas familiares.
- Construir componentes reusables de software y bibliotecas fácilmente extensibles de módulos de software.
- Modificación y extensiones sencillas de las implantaciones de componentes sin tener que recodificar todo desde cero. [KHO90]

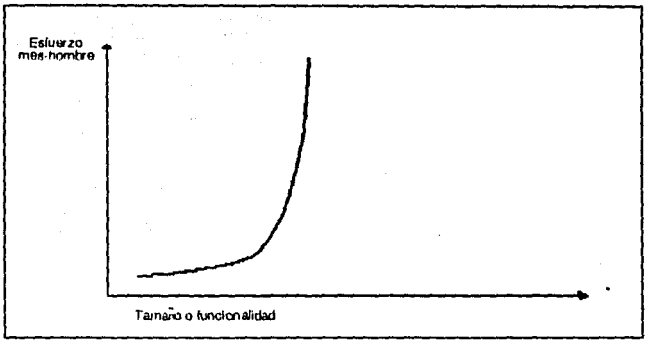


FIGURA 2.1. El esfuerzo para desarrollar un sistema aumenta exponencialmente como una función del tamaño o funcionalidad.

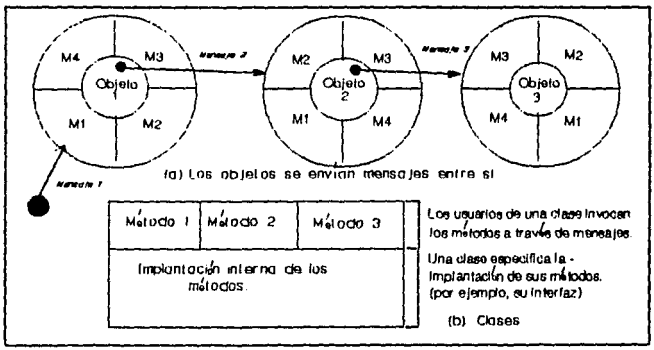


FIGURA 2.2. Objetos (instancias) y clases.

La orientación a objetos pretende satisfacer las necesidades de los usuarios finales así como las necesidades de los desarrolladores de productos de software. Esto se logra por medio de capacidades de modelado del mundo real, ilustrado en las figuras 2.2, 2.3 y 2.4. En la programación orientada a objetos, las entidades centrales son datos (en contraste a los procedimientos o subrutinas). Los datos de un objeto se comunican o se invocan uno a otro mediante el envío de mensajes entre sí. Esto se ilustra en la figura 2.2a. Las colecciones de objetos que responden a los mismos mensajes son implantados a través de clases. Una clase describe e implanta todos los métodos que engloban el comportamiento de sus instancias. Esto se ilustra en la figura 2.2b. La implantación está totalmente oculta o encapsulada dentro de la clase. En otras palabras, las implantaciones pueden modificarse o extenderse sin afectar, de manera alguna, a los usuarios de la clase. Por tanto, las clases contribuyen tanto a la formación de modelos del mundo real y a la extensibilidad del software y a la reusabilidad. Las clases implantan un concepto fundamental en la teoría de la orientación a objetos, los tipos de datos abstractos.

Una clase es como un módulo. Con la orientación a objetos, estos módulos pueden usarse no sólo en diferentes aplicaciones, sino pueden extender o especializar una clase. Esto se logra a través de la herencia. La figura 2.3 muestra tres ejemplos de herencia: una jerarquía de herencia con raíz en Persona, una jerarquía de herencia con raíz en Vehículo y una jerarquía de herencia de objetos de interfaz de usuario con raíz en Ventana. Como ilustra el ejemplo, la herencia de clase proporciona un enfoque muy natural para la organización de información. Adicionalmente, esto permite a los módulos de software compartir el código y su representación (ejemplo obtenido de [HU90])

Cada clase tiene un patrón que describe el estado o estructura de sus instancias. El estado de un objeto puede contener otros objetos. Por ejemplo, un automóvil tiene un motor y una carrocería: el motor de un carro consiste en un sistema de guiado, un sistema de propulsión y un sistema de frenado. El mismo objeto puede ser referenciado o accesado a través de diferentes objetos. El mismo objeto dirección frecuentemente es la dirección de más de una persona; el mismo departamento contiene muchos empleados; el mismo diálogo puede activarse a través de varios botones, etc. La figura 2.4 representa dos empleados, Tim y Terry, que son compañeros de trabajo. Tim y Terry trabajan en el mismo departamento de juguetes. Los objetos Tim y Terry realmente contienen el mismo objeto departamento de juguetes, en lugar de contener copias del mismo objeto, tal como ocurre en los sistemas convencionales. El concepto orientado a objetos que da soporte a organizaciones tan naturales de los espacios de un objeto es llamada identidad de objetos.

A través del uso de estos conceptos orientados a objetos en los diferentes aspectos del software eventualmente podremos lograr una expansión lineal en el esfuerzo como función del tamaño o funcionalidad. Esto se ilustra en la figura 2.5, que muestra que hay una subida inicial en el costo mientras se aprende la tecnología orientada a objetos y se desarrollan los componentes reusables iniciales o las jerarquías de clases iniciales, pero después se logra un crecimiento más lineal en el esfuerzo.

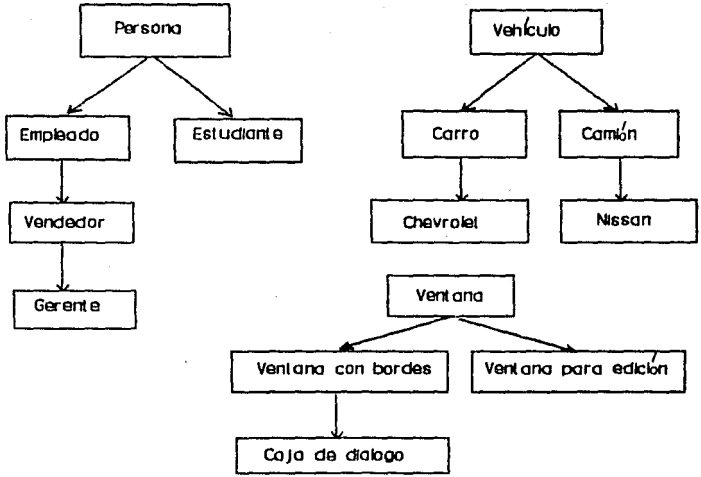


FIGURA 2.3. Ejemplos de jerarquías de clase.

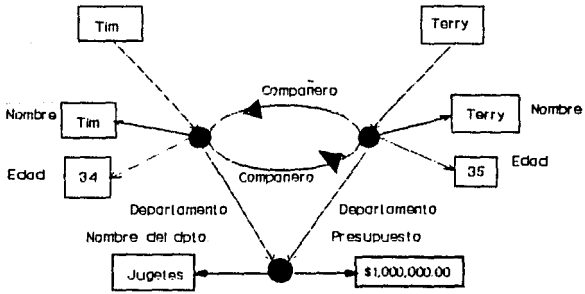


FIGURA 2.4. Objetos que describen a dos socios.

2.1.2. HISTORIA.

El paradigma de la orientación a objetos se basa principalmente en entidades informáticas llamadas objetos y en sus operaciones; estos conceptos no son nuevos en programación; pero el diseño basado en estos conceptos sí. Hacia el inicio de la computación, los ensambladores permitían una representación simbólica de las instrucciones máquina. Algunos de los primeros ensambladores fueron Soap para la IBM 650 (mediados de la década de los 50's) y Sap para la IBM 750. Estos permitían a los programadores el uso de instrucciones máquina (operadores) para manipular datos (operandos).

Aunque básicamente esta es la base del paradigma de la orientación a objetos, el primer hito en el desarrollo de esta teoría fue indudablemente el desarrollo de FORTRAN (mediados de la década de los 50's). Este fue el primer lenguaje de alto nivel desarrollado, e introdujo conceptos importantes en los lenguajes de programación incluyendo abstracciones de control (iteración y salto condicional), estructuración básica de datos (arrays y variables <<tipeadas>> o con un tipo de valor definido).

Empero, a finales de la década de los 50's, surgieron nuevos problemas en el desarrollo de aplicaciones. ¿Era necesario programar en serie, es decir, haciendo un programa grande, con sus variables, estructuras y todo lo demás? ¿Y si se necesitaban tipos de datos no definidos en FORTRAN? ¿Los podrían manejar sus estructuras de control, aunque originalmente no fueron diseñadas para eso? Los lenguajes de programación que sucedieron a FORTRAN (principalmente COBOL y PL/1) trataron de resolver estos conflictos, resumidos principalmente en dos palabras: flexibilidad y tamaño. Fue hasta el desarrollo del lenguaje Algol (principios de la década de los 60's) que surgieron verdaderos cambios en cuanto a diseño de lenguaje. Los diseñadores del lenguaje Algol decidieron proporcionar barreras para separar nombres de variables dentro de segmentos de programa; la implantación Algol 60 dió inicio al uso de los bloques begin..end [PRE87].

Esto resultó ser muy importante desde la perspectiva de la orientación a objetos: dado que una variable solamente existe dentro de los bloques en donde se define, su uso en otros bloques de programa ya no presenta conflicto. Este es el primer intento de proteger o encapsular datos dentro de bloques de programa; la técnica resultó tan exitosa que ahora es usada por una amplia variedad de lenguajes de programación.

A principio de la década de los 60's, se desarrolló un nuevo lenguaje para simulación, Simula-67, que pasó a ser el origen de nuevos conceptos y metodologías en la computación. Este lenguaje tomó el concepto de bloque de Algol y lo llevó un paso adelante al introducir el concepto de objeto; ya no se trataba tan sólo de encerrar un segmento de programa, para facilitar el desarrollo de aplicaciones, los objetos de Simula representaban objetos de la vida real, tenían "existencia" propia y podían (en cierto sentido) comunicarse entre sí durante una simulación. Conceptualmente, un objeto contenía los datos y las operaciones que se les podían realizar a esos datos. Estas operaciones fueron llamadas *métodos*. Simula también introdujo el concepto de clases, que se usan para describir la estructura y el comportamiento que pueden tener un conjunto de objetos. La herencia de clases fue introducida también por Simula, la herencia organiza las clases en jerarquías, permitiendo compartir estructura e implantación. Simula también distinguía entre dos tipos

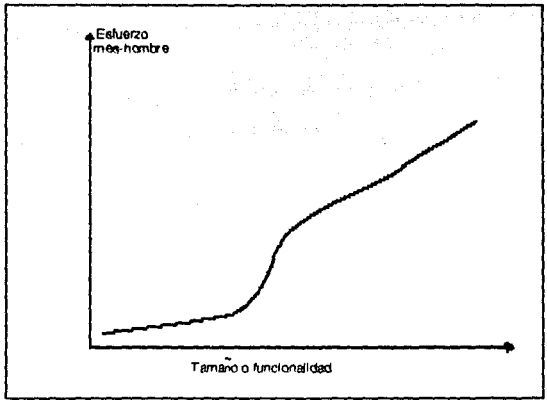


FIGURA 2 5 Por medio de la metodología orientada a objetos, eventualmente se logrará un crecimiento lineal con una mayor funcionalidad.

de igualdad, idéntica y superficial, reflejando la distinción entre interpretaciones de objetos: aquellas interpretaciones basadas en la referencia (identidad) y las basadas en el valor (contenido) del objeto. Por tanto, se puede decir que el lenguaje Simula es la fundación de los lenguajes orientados a objetos y parte de la terminología orientada a objetos.

La revolución estructurada de principio de la década de los 70's también introdujo conceptos importantes. Los diseñadores de lenguajes buscaban implantar el concepto de abstracción de datos. Hasta ese momento, este concepto estaba implantado parcialmente: aunque se pueden definir tipos de datos, no se pueden definir las operaciones que se pueden hacer a esos datos, mucho menos se pueden agrupar operaciones de un cierto tipo de dato en el mismo módulo que la definición de ese tipo, aunque este es uno de los objetivos principales del concepto de la abstracción de datos.

Otro de los conceptos que se introdujeron como una de las reglas de la ingeniería de software, fue el ocultamiento de información, en donde los detalles de la implantación y la representación de una estructura están ocultos y no se pueden accesar directamente por los usuarios del objeto. Este es el fundamento de la modularidad, que facilitó enormemente el desarrollo de aplicaciones grandes y complejas por equipos de trabajo. Este concepto fue presentado en la práctica por lenguajes como Alphard y CLU.

A partir de la segunda mitad de la década de los 70's se desarrolló mucho la teoría matemática que servía de base a la teoría de abstracción de datos. Esto avudó a establecer el concepto de tipos de datos abstractos proporcionando una rigurosa base matemática para usar la orientación a objetos.

En la década de los 80's se desarrollaron mucho los lenguajes Ada y Smalltalk (Ada, a principios de los 80's, Smalltalk fue un prototipo desarrollado en la década de los 70's) que ayudaron a popularizar los conceptos de orientación a objetos. El lenguaje Ada permite la modularización y extensión de estructuras de datos, a través de las diferentes estructuras que tiene incorporadas. Smalltalk es realmente el lenguaje orientado a objetos por excelencia, muchos conceptos orientados a objetos extraídos de Simula y otros prototipos fueron incorporados a este lenguaje. Inicialmente, el Smalltalk fue un proyecto de investigación en el Centro de Investigaciones de Xerox en Palo Alto, California (Xerox Palo Alto Research Center, PARC). Durante la década de los 70's.

Sin embargo, Smalltalk no es realmente un lenguaje; es todo un medio ambiente de programación y una interfaz de usuario altamente interactiva basada en menus. Programar en Smalltalk supone abrir una ventana de trabajo, buscar y extender la jerarquía de clases inicial a través de otra ventana, etc. El programador interactúa con el sistema a través de ventanas desplegadas en la pantalla, dependiendo en la implantación en particular del medio ambiente.

Smalltalk es extremadamente rico en conceptos orientados a objetos. En Smalltalk todo es un objeto, incluyendo clases y tipos base (enteros, números en punto flotante, etc.) Esto significa que en el medio ambiente de Smalltalk, la programación consiste en enviar mensajes a objetos. Un mensaje puede sumar un número a otro número, o puede crear una nueva instancia de una clase, o puede introducir un nuevo método en una clase dada. Smalltalk permanecerá como una poderosa influencia en la programación orientada a objetos. Existe un

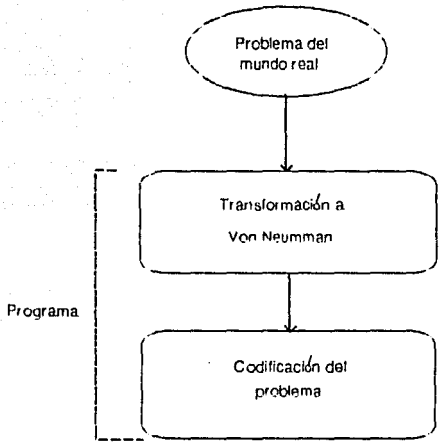


FIGURA 2.6. Esquema de programación convencional.

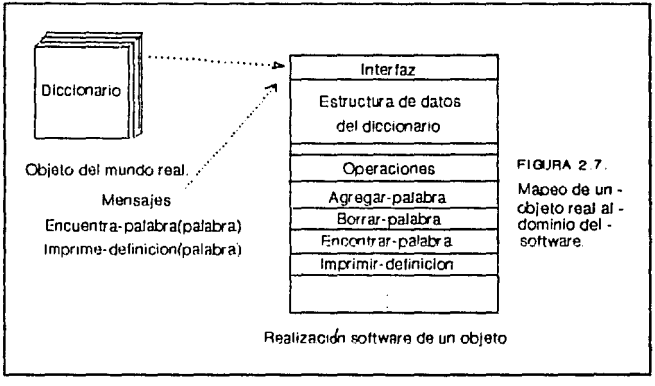


FIGURA 2.7. Mapeo de un objeto real al dominio del software.

diseño y desarrollo continuo de diferentes variaciones y dialectos de Smalltalk, implantados en una amplia variedad de plataformas hardware [KHO90].

2.1.3. DEFINICIONES.

2.1.3.1. El Paradigma orientado a objetos.

La orientación a objetos se puede describir, a grosso modo, como el «conjunto de disciplinas para el desarrollo y modelado de software que facilitan la construcción de sistemas complejos a partir de componentes individuales» [KHO90].

El interés tan grande que genera la orientación a objetos radica en que proporciona mejores conceptos y herramientas para modelar y representar el mundo real tan directamente como es posible. Las ventajas de programación y modelado de datos son muchos. Usando técnicas convencionales, la metodología para generar código a partir de un problema del mundo real consiste en codificar, en primer lugar, el problema y entonces transformarlo a términos de un lenguaje de computadora según la arquitectura definida por Von Neumann. Esto se ilustra en la figura 2.6. Las disciplinas y técnicas orientadas a objetos manejan esta transformación automáticamente, de forma que la cantidad de código necesario tan sólo para englobar el problema y su transformación es minimizada. En efecto, cuando esta metodología se ha puesto a prueba contra estilos de programación más convencionales (procedurales), se han reportado reducciones de código que van desde el 40% hasta un 100% en una gran diversidad de problemas. Los conceptos y herramientas orientadas a objetos están desarrollando tecnologías que permiten expresar mas fácilmente y naturalmente los problemas del mundo real. Las técnicas orientadas a objetos proporcionan mejores metodologías para construir sistemas de software complejos a partir de unidades de software reusables modularizadas [KHO90].

2.1.3.2. Objetos.

Un objeto es un componente del mundo real que esta mapeado sobre el dominio del software. En el contexto de un sistema basado en computadoras, un objeto es típicamente un productor o un consumidor de información o una entidad de información. Entre las ejemplificaciones típicas de objetos, se pueden mencionar: máquinas, comandos, archivos, pantallas, interruptores, señales, cadenas alfanuméricas o cualquier otra persona, lugar o cosa. Cuando un objeto se mapea en su realización en software, su estructura consiste en un conjunto privado de datos y procesos, llamados generalmente métodos, que pueden, en forma legítima, transformar la estructura de datos. Estos métodos consisten en construcciones de control y de procedimiento que se pueden invocar a través de un mensaje -una llamada a un objeto para pedir que realice una de sus operaciones. La realización de software de un objeto se ilustra en la figura 2.7 [PRE87].

2.1.3.3. Propiedades del Objeto.

2.1.3.3.1. Tipificación Abstracta de Datos.

Este es un nombre orientado a objetos para el concepto de abstracción de datos de la ingeniería de software. La tipificación de datos describe un conjunto de objetos (o variables) con la misma representación. Existe un cierto número de operaciones asociadas con

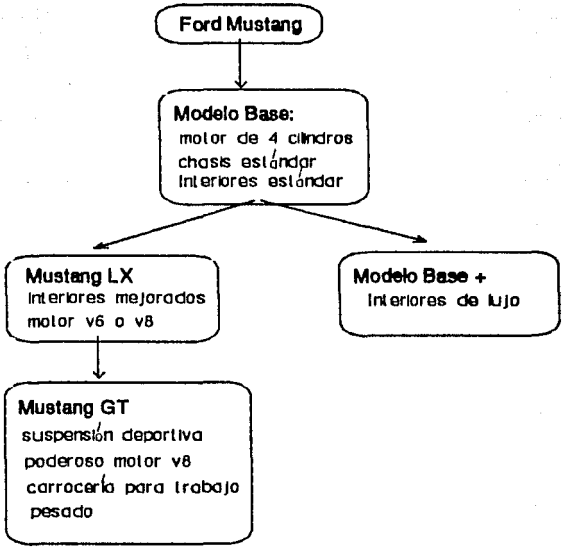
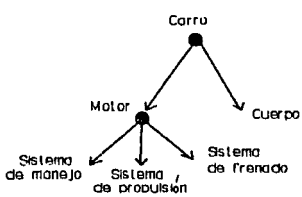


FIGURA 2.8. Jerarquía del Ford Mustang.



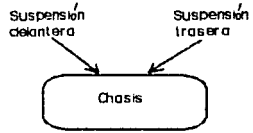


FIGURA 2.9. Componentes y sub_ partes de un auto.

cada tipo de dato (por ejemplo, el tipo entero tiene definida la operación MOD o MODULO, que no está definida para tipos reales). La tipificación abstracta de datos extiende la noción de tipificación de datos a través del "ocultamiento" de la implantación de las funciones definidas por el usuario (mensajes) asociadas con cada tipo de dato (por ejemplo, se puede definir una función MOD para un tipo de datos definidos por el usuario). Los lenguajes que soportan tipificación abstracta de datos proporcionan construcciones para definir directamente estructuras de datos y las operaciones usadas para manipular las ocurrencias (instancias) de las estructuras de datos. Adicionalmente, todas las manipulaciones de las instancias de un tipo de dato se hacen exclusivamente a través de las operaciones asociadas con el tipo de dato.

A continuación se mencionará un ejemplo simple de diseño automotor para ilustrar los conceptos orientados a objetos. Tipicamente, un modelo automotriz tiene muchos modelos relacionados que tienen características y partes en comun. Hay que notar que la operación del auto no implica necesariamente el conocimiento del diseño o la estructura interna. Para manejar un carro, el conductor necesita interactuar solamente con los controles del tablero y los pedales. Básicamente, esto captura la noción de la tipificación abstracta de datos o encapsulación. Un lenguaje que soporte la tipificación abstracta de datos permitirá que las instancias de un tipo de dato sean manipuladas solamente a través de una colección preescrita de operaciones asociadas con ese tipo.

2.1.3.2. Herencia.

El segundo concepto de la orientación a objetos es la herencia. A través de ella, los diseñadores pueden construir nuevos módulos de software (tales como clases) apoyándose en una jerarquia existente de módulos. El comportamiento hereditario permite compartir el código (y por tanto la reusabilidad) entre módulos de software. La representacion hereditaria permite compartir la estructura entre objetos de datos. La combinación de estos dos tipos de herencia proporciona una poderosa estrategia de modelado y desarrollo de software.

Considerese una jerarquía de modelos Ford Mustang, como se ilustra en la figura 2.8. El modelo LX, por ejemplo, hereda la mayor parte de sus características y subpartes del modelo base, pero especializa el interior y el motor. Similarmente, el modelo GT hereda características del LX pero tiene una apariencia y un motor más deportivos. Esta es la esencia de las jerarquias heredadas, donde los tipos de objetos heredan la mayoria de sus atributos de clases más genéricas o de tipos menos especializados.

2.1.3.3. Identidad de Objetos.

El tercer concepto principal orientado a objetos es la identidad de objetos. La identidad es la propiedad de un objeto que distingue cada objeto de todos los demás. Con la identidad de objetos, los objetos pueden contener o referirse a otros objetos. La identidad organiza a los objetos en un espacio de objetos manipulado por un programa orientado a objetos. La tipificación abstracta de datos y la herencia son usadas para modelar y organizar los tipos o clases de objetos.

Para ilustrar los beneficios de la identidad de objetos, se usará un ejemplo de un modelo de carro. (Ver figura 2.9). Cada auto consiste de subpartes: un sistema motriz y un cuerpo. El sistema motriz se puede conceptualizar como un conjunto formado por un sistema de

manejo, un sistema de propulsión y un sistema de frenado. Las subpartes están interconectadas entre si a través de alambres y otros componentes. Por ejemplo, el chasis se comparte y se accesa por los sistemas de suspensión delantera y trasera. Por tanto, la conceptualización de un auto es un grafo (o gráfica) donde las subpartes se accesan o se referencian por otras subpartes. En los lenguajes orientados a objetos, esta representación estructurada como grafo se hace posible a través de la noción de identidad de objetos [KHD90].

2.2. PRACTICA DE LA ORIENTACION A OBJETOS.

2.2.1. ANALISIS Y DISEÑO ORIENTADO A OBJETOS.

2.2.1.1. Introducción.

El uso de conceptos orientados a objetos no es el seguimiento de una tecnología bién definida, sino más bién una actitud. Desde este punto de vista, la programación orientada a objetos es equivalente a la programación estructurada. Los programas están estructurados debido a que tienen ciertos principios de funcionamiento, cierta apariencia y organización, no porque sigan algún conjunto bien definido de reglas. Los beneficios del análisis, el diseño y la programación estructurada se han logrado debido a que los desarrolladores de programas (programadores, analistas, diseñadores, administradores de informática, etc.) han adoptado una nueva actitud respecto a su trabajo.

2.2.1.2. Análisis de Requerimientos.

El término "análisis de requerimientos orientado a objetos" es algo redundante, debido a que es difícil ver cómo hacer un análisis de requerimientos adecuado sin concentrarse en los objetos visibles desde el punto de vista del usuario. Análisis de requerimientos orientado a objetos significa que el análisis estará concentrado en objetos; implica la implantación de un sistema cuya estructura en la solución consista o esté organizada alrededor de objetos. En vez de usar un conjunto totalmente nuevo de herramientas de modelado y técnicas asociadas inventadas exclusivamente para representar al problema desde una perspectiva orientada a objetos, es más útil preguntarse si los modelos existentes son lo suficientemente orientados a objetos.

Otra parte esencial del modelaje orientado a objetos es el uso de diagramas de relaciones de entidades para modelar tanto las partes de una entidad compleja como las relaciones entre entidades dentro del mundo del usuario y de la solución. Los diagramas de relaciones de entidades proporcionan una herramienta excelente para la comunicación entre el dominio y los especialistas en computación. Ayudan en mucho a localizar candidatos a objetos mediante la simplificación y la descripcion clara de las entidades y sus relaciones [BULB9]. En la figura 2.10 se muestra de forma simplificada un diagrama de relaciones de entidades dentro del dominio de un problema.

2.2.1.3. Principios de Diseño.

El diseño descendente (top-down) tiene la siguiente estructura básica: inicialmente se hace un análisis para determinar los procesos y el flujo de datos que implica un sistema. Se especifica la automatización de ciertos procesos; los diseñadores descomponen los procesos en subprocesos más y más pequeños hasta que se crean módulos que hagan una función específica, estos se codifican y se prueban individualmente y entonces se integran y se hace una prueba a todo el sistema. Se completan los ejecutables o biblioteca del sistema y el sistema completo se ensambla y se le hace una prueba adecuada. El proceso es intrínsecamente orientado a procesos. La figura 2.11 ilustra en un diagrama esta metodología de diseño; existen diferentes metodologías de diseño estructurado, sin embargo, cada una está basada en la metodología general de diseño orientado a procedimientos, aunque muchas metodologías usan sus propias formas de diagramación.

Tal como el desarrollo estructurado, el desarrollo orientado a ob-

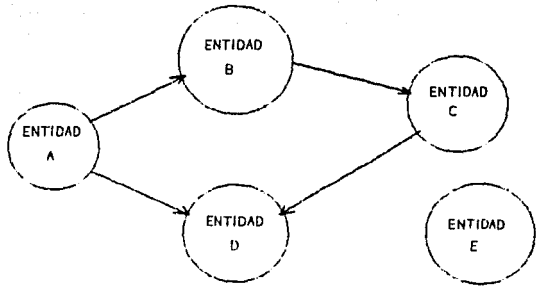


Figura 2.10 Diagrama de relaciones entre entidades.

En este tipo de diagramas, se deben definir las relaciones que tendrá la entidad E respecto a las demas entidades.

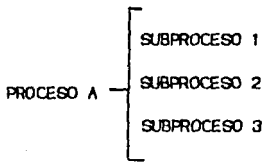


Figura 2.11. Descomposición en subprogramas.

jetos sigue el modelo clásico de cascada: analizar, diseñar, implantar unidades, implantar programas, validar, verificar y mantener. El secreto detrás de los procedimientos orientados a objetos es que inmediatamente después de que se haya completado el análisis, los diseñadores orientados a objetos se enfocan en los datos, no en los procesos [BAIB9].

Para poder llegar a una solución orientada a objetos de un problema, se debe decidir qué objetos deben existir.

Con problemas muy pequeños, la selección de objetos es trivial; esto es, se pueden identificar los objetos a utilizar por simple inspección. Para problemas grandes y complejos la identificación "por inspección" rara vez funciona. Es cierto que algunos objetos corresponderán a entidades de usuario reconocibles en el dominio del problema. Empero, no todas las entidades son obvias ni pueden descubrirse intuitivamente; además, las entidades de usuario no son las únicas fuentes de objetos.

Desde el punto de vista de un programa orientado a objetos, un objeto es un conjunto privado de datos junto con sus operaciones, se dice que los datos son privados porque deben ser válidos solamente dentro del ámbito de acción del objeto, es decir, las operaciones que forman el objeto son las únicas que debieran tener acceso a dichos datos; ninguna función o proceso externo debe tener acceso a ellos. Para propósitos de diseño, un objeto se caracteriza enteramente por sus operaciones. Esto es, dentro de la etapa de diseño, el uso de objetos se hará tan sólo considerando las operaciones que serán capaces de realizar, sin importar los datos que tiene el objeto. Por ejemplo, no importa si un objeto de factura tiene una variable local que tenga el total de la factura o si el total es calculado a partir de datos individuales integrantes del objeto; el total no necesariamente tiene que ser una propiedad almacenada del objeto. Lo único que se necesita saber es que el total de la factura estará disponible cuando se invoque el objeto. Por tanto, cualquier metodología propuesta para el diseño orientado a objetos debe enfocarse a la identificación y especificación de las operaciones con objetos [BULB9].

Las figuras 2.12a, 2.12b y 2.12c muestran los diagramas de descomposición de un sistema para el control de un automóvil (en forma simplificada), se leen y se calculan distintas velocidades y estados y el sistema responde con una señal de abrir/cerrar (ajustar) las válvulas de combustible. La figura 2.12a muestra un diagrama del flujo de datos necesario en el sistema. En la figura 2.12b se han usado las técnicas propuestas por Yourdon-Constantine para descomponer el sistema usando un modelo entrada-proceso-salida, en módulos que denotan las funciones principales del proceso en general. De esta forma, primero se consideran las funciones principales que tratan con las entradas al sistema (Obtener Velocidad Deseada, Obtener Velocidad Actual, Obtener Estado de los Frenos); después se considera la función principal para el procesamiento de estos datos (Calcular Estado de las Válvulas). Finalmente se considera la función principal que representa la salida del sistema (Poner Ajustes de las Válvulas).

La figura 2.12c muestra un diagrama enfocado a la orientación a objetos. Dentro de este diagrama las consideraciones hechas son totalmente distintas. En vez de factorizar el sistema en módulos que denotan operaciones, el sistema se estructura alrededor de objetos que existen en nuestro modelo de la realidad. Extrayendo los objetos

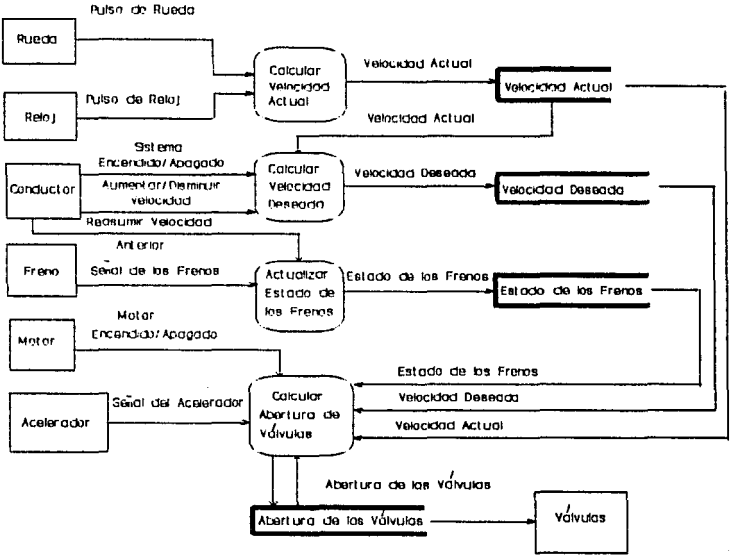


Figura 2.12. Sistema para el Control de un Automóvil.
(a) Diagrama de flujo de datos

implicados en el diagrama de flujo de datos anterior (la función Obtener Estado de los Frenos implica la existencia de un objeto llamado Frenos, por ejemplo), se genera la estructura mostrada en la figura 2.12c [BOOS87].

2.2.1.4. Diseño Orientado a Objetos.

Un diseño orientado a objetos es aquel en el que cada uno de los módulos mayores corresponde a una entidad, ya sea en el mundo del usuario o en el de la implantación de la solución del problema. El primer paso en esta estrategia es la construcción de una lista parcial de objetos candidatos, llamados así porque se requerirán algunas iteraciones para llegar a su definición final.

La mayoría de los objetos candidatos se derivan directamente del modelo propuesto por el análisis. Los objetos se refinan agregando más objetos o particionando y combinando los ya existentes. También se refinan las operaciones definidas para cada objeto. Este paso ayuda a la creación de objetos reusables. Uno de los objetivos principales es localizar las oportunidades de definir operaciones de alto nivel para objetos existentes. El reconocimiento y la implantación de operaciones de alto nivel causa que el diseño del resto del sistema sea más simple [BUL89].

La derivación de objetos se hace principalmente durante las etapas de análisis y diseño, y de forma secundaria mientras se escribe el código fuente en algún lenguaje orientado a objetos. Durante el análisis se construye un modelo del problema de forma tal que se asegure que se entendió el problema. Durante el diseño se construye un modelo de la solución del problema. En esta etapa se identifican diferentes tipos de objetos, incluyendo entidades que existen solamente en el sistema de computación: hardware tangible, objetos de software tales como paquetes comprados, depósitos de datos e índices de archivos; interfaces presentadas en la solución propuesta e interfaces a otros procesadores dentro de un sistema distribuido.

El modelo de diseño debe servir de ayuda para enfocar los diferentes tipos de objetos que se tendrán que derivar para representar estas entidades específicas a la solución. El modelo del diseño pone énfasis en los objetos y sus representaciones. Llegado a este punto, la solución deberá, por lo menos, estar representada en términos de objetos, sus operaciones y datos, junto con una vista general de la comunicación entre objetos.

El análisis realizado con cada sistema es único, en el sentido de que cada análisis necesita objetos de muy diversas estructuras y comportamientos. En el análisis hay más que los datos que se necesitan. También se pueden descubrir y documentar las acciones que se necesitan. Cada acción requerida por el usuario causará que se asignen operaciones a uno o más objetos. El conjunto mínimo de operaciones para cada objeto consiste exactamente de aquellas operaciones que cumplan los requerimientos del usuario [BUL91].

2.2.2. PROGRAMACION ORIENTADA A OBJETOS.

Una vez que se ha terminado la fase de diseño, puede empezar la fase de codificación o programación del sistema. La programación orientada a objetos, tal como el diseño orientado a objetos es más una actitud que una serie de reglas. Es un esfuerzo para implantar una clase mientras se preservan los conceptos del diseño tales como la

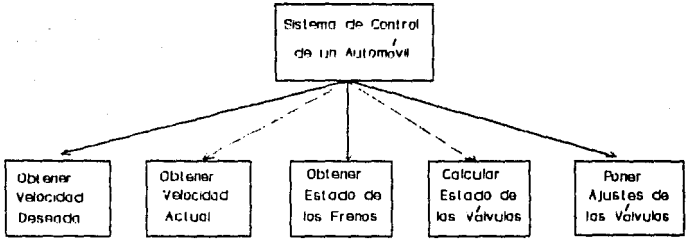


Figura 2.12 (b). Descomposición funcional.

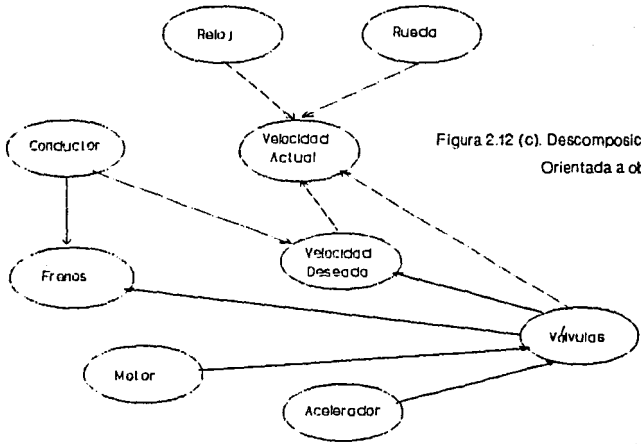


Figura 2.12 (c). Descomposición Orientada a objetos.

modularidad, encapsulación de datos y ocultamiento de información.

Ningun diseñador puede tomar en cuenta por adelantado todos los aspectos de diseño, por lo que los programadores tienen que hacer un número considerable de decisiones de implantación. La base para tomar estas decisiones son las consideraciones de diseño: mantener la modularidad, la encapsulación de datos y el ocultamiento de información. Estos tres conceptos materializan los beneficios de la revolución orientada a objetos y son tan importantes durante la fase de programación como lo eran en la fase de diseño.

Uno de los fundamentos de las metodologías estructuradas y orientada a objetos es la modularidad; bajo un enfoque orientado a objetos, <<modularidad>> es sinónimo de <<clase>>. Cada clase es tratada como una unidad independiente; de esta forma se define, se diseña, se construye y se prueba. La única razón para considerar otra unidad mientras se diseña y se construye una clase, es para asegurarse que existen todas las interfaces requeridas.

Los módulos a usar se pueden implantar de diferentes formas dependiendo del lenguaje usado; lo más apropiado es usar las facilidades del lenguaje (o del compilador) para mantener la modularidad. De esta forma se puede implantar cada clase como un solo archivo independiente, de esta forma, si se necesita trabajar con un método de una clase, siempre se sabrá donde encontrarla.

La encapsulación de datos se maneja principalmente durante la fase de diseño; empero, puede volverse un aspecto de la programación. Frecuentemente, lo que inicialmente parecia un objeto, en realidad está compuesto por sub-objetos. Los programadores y los diseñadores de un sistema deben reunirse para determinar la encapsulación apropiada y las interfaces de los objetos que se van a crear.

Muchos programadores definen una estructura de datos en una sola clase, cuando se pierde flexibilidad y sencillez al meter todos los datos en una sola estructura, entonces se requieren dos o más clases. Este enfoque tan simplista puede no funcionar para todas las instancias; sin embargo, la mayoria de los datos a usar para representar los objetos componentes del sistema pueden implantarse fácilmente a través de una estructura. Los métodos orientados a objetos pueden extenderse fácilmente a aplicaciones de bases de datos simplemente mapeando todas las tablas de datos para crear nodos en la jerarquia de clases.

El ocultamiento de información es el concepto más difícil de reforzar en la codificación. La idea es esconder de otras partes del programa o sistema la implantación de aquellas decisiones que cambiarán con más probabilidad.

La estructura de datos y su acceso, su almacenamiento y los procedimientos que modifican estas estructuras, son las entidades que más frecuentemente cambian dentro de un programa o sistema. El ocultamiento de la implantación de esta información se vuelve, entonces, aspecto crucial de la programación de un sistema.

En síntesis, el ocultamiento de información significa que la lógica del acceso, almacenamiento y rutinas para modificación de estructuras de datos debe ocultarse de rutinas que no estén involucradas específicamente con las estructuras. Debido a que la lógica de cualquier proceso depende de la estructura que se esté manipulando, la estructura de un objeto debe ocultarse también. El objeto y sus métodos deben ser lo que comúnmente se denomina caja negra [BAI89].

2.2.3. LENGUAJES DE PROGRAMACION.

El paradigma orientado a objetos tiene como objetivo final el poder hacer que las computadoras resuelvan un problema sin tener que introducirles una secuencia de instrucciones especificas para esa tarea; es decir, usar computadoras sin necesidad de programarlas. En la actualidad, el objetivo a alcanzar por las técnicas orientadas a objetos es el poder definir diversos tipos de objetos y lograr la construcción y puesta a punto de un sistema tan sólo indicando a la computadora que interrelacione los objetos necesarios; en algun caso en particular, que herede caracteristicas de objetos ya establecidos para formar nuevos objetos que constituyan el conjunto para construir una solución.

Aunque falta mucho desarrollo para conseguir el objetivo de la «computación sin programar», ya se han dado pasos en la dirección correcta. En capitulos anteriores se mencionó al lenguaje Smalltalk como uno de los pilares actuales de los conceptos orientados a objetos. Tipicamente, los programadores que trabajan en Smalltalk no siguen el modelo clasico editar-compilar-depurar, es decir, no proceden cargando en su computadora un editor de texto y escribiendo el código; aunque Smalltalk permite esta metodologia de desarrollo como cualquier otro lenguaje, el seguir este modelo de desarrollo desperdicia los recursos de Smalltalk. El medio ambiente Smalltalk está compuesto de objetos interrelacionados que son instancias o incorporaciones de clases.

El usuario tiene acceso a estas clases y objetos y a sus definiciones que forman el código fuente en el que se basará el sistema y que conforman el punto de partida de todo desarrollo futuro. Por tanto, la programación en Smalltalk tipicamente se realiza mediante la expansión del medio ambiente, no mediante la generación de un cuerpo de código fuente externo que usa un número determinado de funciones del lenguaje.

Smalltalk posee un conjunto de primitivas de lenguaje a partir de las cuales se construye el medio ambiente, pero la relación entre el lenguaje y el medio ambiente de desarrollo es más estrecha en Smalltalk que en otros lenguajes. En Smalltalk, el código se puede probar corriendolo directamente en el medio ambiente de desarrollo en vez de compilarlo por separado (como en la mayoría de los lenguajes) y correrlo después bajo el control del sistema operativo [SUY90].

A pesar de todas las ventajas que ofrece Smalltalk, como lenguaje y como medio ambiente de desarrollo, también tiene fuertes desventajas que, en opinión de algunos programadores, pesan más que las ventajas ofrecidas. La principal es que Smalltalk es radicalmente diferente a los conceptos de programación usuales. Otra desventaja en el uso de Smalltalk es de caracter industrial; Smalltalk es un lenguaje interpretado, la ejecución de sistemas grandes requiere demasiado tiempo de procesador, aunque con el desarrollo y la comercialización de nueva tecnologia electrónica, el uso de un lenguaje interpretado no significa necesariamente que la ejecución sea prohibitivamente lenta.

El lenguaje Pascal ha sido un lenguaje ampliamente usado por sus caracteristicas: es sencillo, concreto, estable y permite implantar con elegancia todas las técnicas de programación y, recientemente (en

los últimos años), de la orientación a objetos.

El compilador Turbo Pascal de Borland ha incorporado muchas características muy útiles y poderosas al lenguaje estándar. Una de las características que se han incorporado (a partir de la versión 5.5) es la orientación a objetos. Turbo Pascal ofrece muchas de las ventajas que facilita el medio ambiente Smalltalk: el desarrollo de un sistema se puede hacer dentro de un ambiente integrado, o siguiendo el modelo editar-compilar-depurar; dadas sus características de modularidad y orientación a objetos, en Turbo Pascal se puede seguir un desarrollo tan sólo seleccionando y extendiendo objetos; y debido a que todas estas características son adicionales al estándar, Turbo Pascal puede seguir usandose como un lenguaje orientado a procedimientos, o usarse como un lenguaje totalmente orientado a objetos, o como un lenguaje híbrido: estructurado en parte y con características orientadas a objetos.

El artículo Objects, Functions and Program Extensibility de Larry L. Constantine (uno de los miembros más distinguidos de la comunidad informática) propone integrar los dos paradigmas dentro del diseño de un sistema. De esta forma, en vez de aprender una sintaxis totalmente nueva, el uso de un medio híbrido requiere solamente que los programadores se deshagan de ciertos hábitos establecidos y aprendan a usar algunas nuevas herramientas, sin deshechar totalmente la práctica que se tiene con el lenguaje. Los resultados y beneficios de este intercambio de costumbres no implican meramente una modificación en las prácticas de programación, sino una dirección completamente nueva en el desarrollo general de un sistema.

CAPITULO 3

Teoría de Bases de Datos.

3.1.ANTECEDENTES DE LOS SISTEMAS DE ADMINISTRACION DE BASES DE DATOS (DBMS).

3.1.1. Definiciones de Base de Datos.

Estrictamente hablando, una base de datos es solamente uno o varios archivos que contengan datos. En forma práctica este término hace referencia a muchos otros conceptos: una base de datos presupone o implica la existencia de un programa que pueda manipular los datos contenidos en el archivo o los archivos existentes; de esta manera, se debe hablar de un Sistema de Base de Datos, porque se manejan programas y archivos formando un sistema para lograr un objetivo coherente. De una manera mas formal: "Una base de datos [o un Sistema de Base de Datos] es una colección de información usualmente organizada y ordenada para ser utilizada en una aplicación o con algun objetivo en particular" [CHAB9]. La estructura básica de una base de datos, entendida como un sistema de archivos de información, es la siguiente: el componente basico son los archivos, que en terminología de bases de datos son llamados tablas. Una tabla es una colección de información organizada en renglones (registros) y columnas (campos). Para que la recopilación de información se haga en forma coherente, los datos que se capturan y que pertenecen a una sola entidad se guardan en un renglón de la tabla con una relación lógica llamada eneada. Es decir, una eneada toma al conjunto de renglones en una o varias tablas como una serie de elementos dato relacionados y heterogéneos (una eneada es capaz de reconocer un conjunto de renglones de varias tablas debido a que establece vínculos lógicos entre los elementos). Esta terminología definida en bases de datos es muy importante, puesto que es considerada estandar y se usará intensivamente en el siguiente capitulo.

Un Sistema de Base de Datos se basa, fundamentalmente, en dos cosas: una serie de archivos que contienen los datos y un programa que sepa como accesar esos datos. El programa en cuestión debe saber cómo leer los datos, cómo escribirlos, la forma en que los va a presentar al usuario, etc. Un programa con estas caracteristicas es llamado Administrador, Manejador o Gestor de Bases de datos; en forma más general, un Sistema Manejador de Bases de Datos (en adelante llamado DBMS) es la combinación de programas y tablas que se utilizan conjuntamente para el manejo de datos [CHAB9]. Este es el concepto que engloba en realidad el término <<base de datos>>. Para que un DBMS pueda ser considerado como administrador conveniente para el almacenamiento de grandes cantidades de datos, debe mostrar las siguientes caracteristicas:

- Rápido acceso para la recuperación de información.
- Actualización conveniente.
- Economía de almacenamiento.

La capacidad para representar estructuras de información del mundo real, la confiabilidad, la protección de la privacia y el mantenimiento de integridad son criterios importantes que se deben evaluar en la organización de cualquier sistema de archivos [CHAB9].

3.1.2. Evolución de las bases de datos.

Los antecesores de los DBMS eran simples rutinas generales de archivos. Estos sistemas de manejo de archivos eran capaces de

desempeñar discretamente algunas operaciones comunes como la ordenación, mantenimiento de archivos y generación de reportes.

Durante las décadas de los 50's y los 60's se empezaron a desarrollar algunos productos de definición de datos; estos productos permitían definir la estructura de bases de datos que se podían accesar por múltiples usuarios.

En 1969, se propuso una definición para un Lenguaje de Descripción de Datos (DDL - Data Description Language) y un Lenguaje de Manipulación de Datos (DML - Data Manipulation Language) para bases de datos, esto sirvió como base para la fundación de los sistemas de redes para administración de bases de datos.

Los primero sistemas para la administración de bases de datos se basaban en la teoría de relaciones jerárquicas y de redes entre los datos. La presentación de red consiste en tipos de registro y relaciones de uno a muchos entre estos tipos de registro. Por ejemplo, una organización puede tener un tipo de registro para almacenar los datos de un empleado, donde cada empleado tiene un nombre, dirección, número de seguro social y un salario. La organización puede tener también un tipo de registro departamento que contenga el nombre del departamento y el presupuesto para ese departamento. La representación de red incluye una relación uno a muchos entre los tipos de registro departamento y empleado, indicando que cada departamento puede tener muchos empleados.

El modelo de red permite que un tipo de registro tenga más de una relación. Un modelo menos general plantea un tipo de relaciones jerárquicas entre registros, estructuradas en árbol; este tipo de relaciones es la base para el modelo jerárquico. El modelo de datos jerárqico permite que un tipo de registro tenga solamente una relación como "padre" y solamente una relación como "hijo".

Los modelos de datos jerárquico y de redes eran principalmente, "navegacionales"; un usuario empieza desde un registro padre o propietario y navega a través de los miembros de una relación a través de las construcciones get next, get first o get last (construcciones ir al próximo, ir al principio o ir al final). Además, las relaciones propietario/miembro (para el modelo de red) o la relación padre/hijo (para el modelo jerárquico) se almacenaban en forma explícita en los registros de la base de datos. En forma más específica, las implantaciones de DBMS jerárquicas y de redes no tenían una independencia física de los datos. Esto significaba que el punto de vista del usuario de las bases de datos reflejaban la manera en que estaban organizados, almacenados y la forma en que se accesaban los datos según fuera el medio físico de almacenamiento. En algunos casos, se requería que el usuario o el administrador del sistema de manejo de las bases de datos (Data Base managment system Administrator, DBA) especificara los detalles para el emplazamiento de registros, areas de almacenamiento, ordenación de registros, locaciones de registro, etc. Además del problema que implicaba tener que dar especificaciones técnicas (y asegurarse que las especificaciones dadas eran correctas), este enfoque limitaba seriamente la extensibilidad, la facilidad de mantenimiento, la reusabilidad y la portabilidad de las aplicaciones desarrolladas a partir de los sistemas de manejo de bases de datos basados en estos modelos.

A principio de la década de los 70's, Ted Codd propuso el modelo

relacional de datos para proporcionar más flexibilidad en la organización de bases de datos grandes, y para paliar algunos problemas de los modelos anteriores. El modelo relacional para DBMS prevalece sobre el uso de los otros dos modelos principales debido a varias ventajas inherentes. La primera de esas ventajas es que los lenguajes relacionales de consultas como el SQL (Structured Query Language - Lenguaje Estructurado para Consultas) son mucho más declarativos que los lenguajes navegacionales de los primeros modelos; esto significa que el usuario especifica lo que se quiere de una base de datos en un estilo declarativo de alto nivel de programación, diciendo qué es lo que se va a accesar de la base de datos, y no como accesarlo. El modelo relacional es simple y elegante; la teoría subyacente está basada en los conceptos matemáticos bien fundados y entendidos del algebra relacional y el cálculo de predicados de primer orden. El algebra relacional consiste tan solo de unas pocas operaciones: operaciones entre conjuntos (unión, intersección, diferencia, producto cartesiano) y operaciones relacionales (selección, proyección, reunión, división).

A finales de la década de los 80's, casi todos los DBMS comerciales estaban basados en los modelos jerárquico, de red o relacional. Las bases de datos relacionales siguen expandiendo su popularidad y su uso. Aun así, han habido varias propuestas alternativas sobre modelaje de base de datos; muchos de estos modelos post-relacionales de datos fueron desarrollados como prototipos en laboratorios de investigación y nunca fueron comercializados.

Una de las primeras propuestas alternativas para el modelaje de base de datos fue el modelo semántico. La motivación para el modelo semántico (y para la mayoría de los modelos de datos) es muy similar al objetivo de la orientación a objetos: modelar el mundo real tan cercanamente como sea posible. El enfoque de relaciones entre entidades es una técnica ampliamente aceptada para el modelaje de datos. Una entidad es un objeto o cosa que existe y que puede distinguirse de otras entidades; una entidad puede ser una persona, una institución, un vuelo, etc. Las entidades se describen a través de atributos o propiedades. En términos de construcciones regulares de bases de datos, las entidades conforman registros mientras sus atributos se representan mediante campos de esos registros.

Los modelos semánticos se usan principalmente como herramientas de diseño de bases de datos; cuando está listo este esquema semántico, se mapea en un esquema relacional usando el lenguaje de definición de datos de alguna base de datos relacional (tal como el DDL de SQL). El usuario puede entonces recuperar y actualizar los datos almacenados por medio del esquema usando un lenguaje relacional de manipulación de datos (tal como el DML de SQL). Pero esto es inconveniente y no natural; han habido intentos de incorporar capacidades de manipulación de datos en el modelo semántico, a través de las relaciones funcionales.

El enfoque semántico para modelaje de datos no fue el único que trató de agregar más semántica a los modelos tradicionales; hubo más modelos de datos que han tratado de extender incrementalmente el modelo relacional para permitir una mayor flexibilidad, mientras se conserva una sólida fundación teórica.

El espacio de los objetos en el modelo relacional consiste de una

colección de tablas planas. Cada tabla es un conjunto de filas o renglones (tuplas); los valores en columnas de cada fila (atributos de las tuplas) solamente pueden ser instancias de tipos atómicos de datos tales como enteros, reales o cadenas de caracteres. La representación de tablas planas es conocida como la primera forma normal. Otro modelo post-relacional es el llamado modelo de objetos complejos. Los modelos de objetos complejos intentan relajar las restricciones de la primera forma normal; los modelos de objetos complejos son conocidos también modelos de no-primera forma normal, modelos de relaciones anidadas, modelos de conjunto y tuplas o modelos de datos basados en la lógica con términos complejos.

Con un modelo relacional anidado, el usuario podrá tener atributos valuados en la relación (esto es, basado en un conjunto de tuplas) y, por tanto, podrá representar, almacenar o recuperar directamente un conjunto de tuplas. Se pueden hacer modelos más generales para objetos complejos (incluyendo el anidamiento de objetos arbitrarios) construyendo espacios de objetos encima de una colección tipos atómicos básicos, usando dos constructores de objetos: conjuntos y tuplas.

Los modelos de objetos complejos post-relacionales mencionados hasta ahora, permiten solamente espacios de objetos estructurados en árbol; aunque esto está bien fundamentado conceptualmente, estos modelos no permiten que un objeto sea un sub-objeto de padres múltiples, tampoco soportan espacios de objetos estructurados en grafos. Para que se presentan estas características, los modelos deben soportar el concepto orientado a objetos de identidad de objetos.

3.2. MODELOS DE DESARROLLO.

3.2.1. Modelo JerArquico.

Este modelo se comprenderá mejor si se considera lo que es una jerárquia o árbol (los términos se usarán indistintamente). En la figura 3.1 se muestra un árbol abstracto o jerarquia, el cual sólo es un arreglo de puntos denominado nodos (numerados del 1 al 11) conectados por ramas o aristas (lineas rectas de la figura) con las restricciones que siguen. Existe un nodo especial denominado raiz que siempre se coloca en la parte superior del diagrama. En la figura 3.1 el nodo 1 es la raiz del arbol. Las ramas del árbol crecen hacia abajo a partir de la raiz; cada nodo es capaz de ramificarse en varios en el siguiente nivel descendente. Como resultado, todo nodo (con excepción de la raiz) está conectado hacia arriba con un solo nodo. Por ejemplo, el nodo 3 está conectado hacia arriba sólo con el 1; el nodo 9 está conectado hacia arriba (inmediatamente) unicamente con el 6.

Los nodos que descienden en forma directa de un determinado nodo se denominan como sus nodos hijos; por el contrario, cada uno de éstos tiene como nodo padre a aquél del que provienen. Por ejemplo, en la figura 3.1, los nodos 2, 3 y 4 son los hijos del nodo 1. El nodo 2 es el padre de los nodos 5 y 6. Podria hablarse de todos los nodos que están debajo de un nodo dado como sus descendientes. De esta forma, los descendientes del nodo 1 son los nodos 2 al 11. Un nodo sin ramas que se extiendan hacia abajo (por ejemplo, nodos 3, 5 y 8) se denomina hoja o nodo terminal, para distinguirlos del nodos como el 2, 4 y 6 que se denominan nodos no terminales o nodos intermedios. Aquellos nodos que tienen el mismo padre (como el 5 y el 6) se denominan gemelos o hermanos [GIL88].

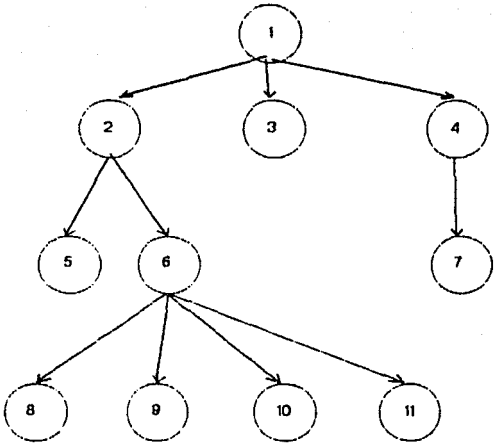
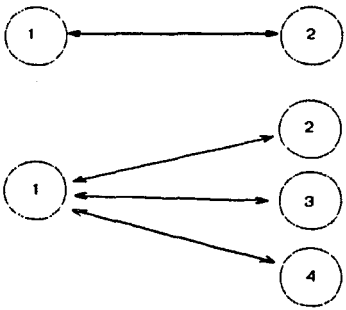


Figura 3.1. Arbol o jerarquía.



a. Relación uno a uno.

b. Relación uno a muchos.

Figura 3.2. Tipos de relaciones entre entidades.

3.2.2. Modelo de Red.

El modelo de red esta basado en una estructura muy parecida a los árboles del modelo jerárquico. La diferencia principal tiene que ver en como están relacionados los nodos, mientras que en el modelo jerarquico los nodos solamente podían tener relaciones uno a uno (es decir, un nodo podia estar relacionado con otro, ya fuera como padre o como hijo, ver figura 3.2(a)), en el modelo de red un nodo puede estar relacionado con otros muchos, (ver figura 3.2(b)). Esto, sin embargo, no disminuyó el caracter navegacional del modelo de red; todavía se tenían que recorrer muchas aristas y visitar diferentes nodos para poder accesar la información deseada. El modelo de redes proporcionó flexibilidad a este tipo de navegación: se podía viajar a cualquier nodo desde cualquier otro nodo situado en otro lugar de la red; pero esta flexibilidad también trajo el potencial de poder crear una estructura pesada y con demasiadas ramificaciones entre los nodos.

La estructura de datos conocida como red, llamada también grafo, es un tipo de estructura muy general: en forma abstracta, la estructura se puede definir como un conjunto de puntos o nodos interconectados de alguna forma. La conexión entre dos puntos se conoce como arista. Esta definición coincide con el concepto de árbol; en realidad, el concepto de red engloba al concepto de árbol, un árbol es un tipo limitado de red.

La figura 3.3 muestra una ejemplificación de una red, con este diagrama se definirán algunos términos. Dos puntos son adyacentes si está conectados por una arista; de esta forma, los puntos 1 y 3 son adyacentes de acuerdo a la figura 3.3. Una arista incide en un punto si lo toca; la arista I incide con el punto 6. El grado de un nodo es el número de aristas que inciden en él; el grado del nodo 5 es 4 debido a que lo tocan las aristas F, G, H e I. Una ruta es un conjunto de aristas conectadas en serie, como las aristas A, C y J, que conectan a los nodos 1 y 6. Un ciclo es una ruta que empieza y termina en el mismo punto; la ruta de las aristas C, D, F y H es un ciclo porque empiezan y terminan en el nodo 2.

3.2.3. Modelo Relacional.

3.2.3.1. Fundamentos.

El modelo relacional se basa en la abstracción matemática de un conjunto: una estructura que puede contener un número arbitrario de elementos y que no tienen un orden determinado entre si (fig. 3.4). De esta forma, cualquier elemento o nodo se puede accesar en forma directa, sin tener que pasar por otros nodos o tener que recorrer una ruta determinada de aristas. Dado que el modelo relacional se basa en la teoria de conjuntos, la base subyacente tiene un firme cimiento matemático para las operaciones en el conjunto. Algunos teóricos, notablemente Ted Codd, ayudaron a la adaptacion de este tipo de abstracciones matemáticas a representaciones informáticas.

Pero el modelo relacional no solo tiene la ventaja de presentar una estructura más conveniente para el almacenamiento y manejo de datos; la teoria matematica también define un cierto numero de operaciones entre conjuntos: lo que se conoce como relaciones.

3.2.3.2. Algebra relacional.

El algebra relacional está definida como la serie de operaciones que operan sobre conjuntos. En terminologia informática, se puede definir lo siguiente: el álgebra relacional es un conjunto de opera _

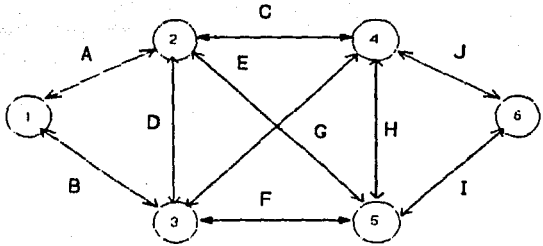


Figura 3.3. Estructura de red.

ciones sobre las relaciones o tablas. Cada operación toma uno o más tablas como sus operandos y produce otra tabla como resultado.

Codd definió originalmente un conjunto de operaciones y demostró que estas operaciones eran completas. Las operaciones del Álgebra relacional son las siguientes:

1. Unión.
2. Intersección.
3. Diferencia.
4. Producto Cartesiano.
5. Selección.
6. Proyección.
7. Reunión.
8. División.

Unión. La unión de dos tablas A y B es una tercera tabla C tal que todas sus eneadas o tuplas pertenecen a A o a B o a ambas [fig 3.5(a)].

Intersección. La intersección de dos tablas A y B da como resultado el conjunto de todas las tuplas o eneadas que pertezcan a A y a B a la vez [fig 3.5(b)].

Diferencia. La diferencia de dos relaciones o tablas A-B es el conjunto de todas las tuplas que pertenecen a A, pero no a B [fig 3.5(c)].

Producto Cartesiano. El producto de dos tablas A X B resulta en la tabla C de forma que las tuplas o eneadas de C están formadas por la concatenación (o producto) de cada tupla de A con cada tupla de B [fig 3.5(d)].

Selección. La selección es una operación de una relación o tabla tal que la tabla resultante es un subconjunto de la tabla original cuyas tuplas cumplen con una condición específica [fig 3.5(e)].

Proyección. La proyección de una tabla A es una tabla C tal que todas las tuplas de C son las mismas tuplas de A, aunque las tuplas de C tienen solamente ciertos atributos de la tuplas de A. Es decir, la proyección filtra ciertos atributos o campos de A [fig 3.5(f)].

Reunión. Esta operación requiere que las dos tablas en las que se va a hacer la operación compartan algún atributo en sus tuplas. La operación tabla A reunión tabla B da como resultado otra tabla C tal que la tuplas de C tienen los mismos atributos o campos que la tabla A, excepto el atributo que compartía con la tabla B. Este campo o atributo se sustituye por los otros campos en B [fig 3.5(g)].

División. Esta es una operación que se hace entre dos tablas siempre y cuando las eneadas o tuplas de éstas tengan un atributo en común. La división de B en A es otra tabla C tal que las tuplas de C cumplan con alguna condición de las tuplas de B, pero sus campos son los mismos que los campos de la tabla A, exceptuando el campo que se tenía en común con la tabla B [fig 3.5(h), CHA89].

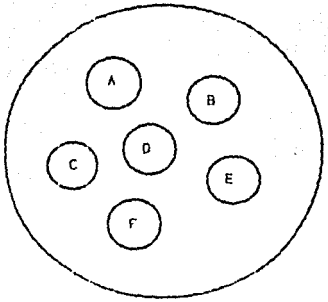


Figura 3.4. Conjunto con elementos

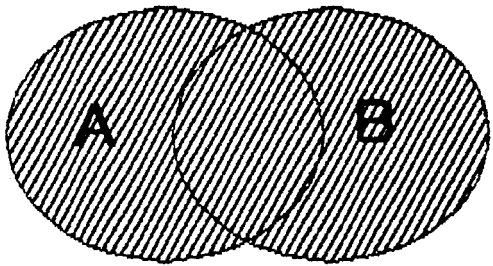


Figura 3.5 (a). Union de conjuntos.

3.2.4. Otros Modelos.

3.2.4.1. Modelos de Objetos Complejos.

Los modelos de objetos complejos son una evolución de los modelos relacionales, estos modelos permiten representar abstracciones de datos dentro de las tuplas. Los modelos relacionales solamente permitían que las tuplas o registros contuvieran una serie de atributos o campos basados en los tipos de datos atómicos o predefinidos dentro del lenguaje de definición de datos (sea un lenguaje de tipo general o un sistema de administración de datos); los modelos de objetos complejos permiten que los atributos o campos de las tuplas o registros sean conjuntos de otros atributos de tipo atómicos (definidos en el sistema) o de tipo definido por el usuario.

Este modelo contiene/permite una o mas de las siguientes extensiones:

1. Conjuntos de valores atómicos (conjuntos de cadenas de caracteres, de enteros, de números de punto flotante, etc.)
2. Atributos con valor basado en las tuplas.
3. Conjuntos de tuplas (relaciones anidadas).
4. Constructores generales de conjuntos y tuplas.
5. Identidad de objetos.

Considerese el ejemplo de la dirección de una persona. Una dirección consiste del nombre de la calle, el número de la casa, ciudad, estado y código postal. No es posible imaginar esta estructura como un conjunto de valores atómicos; cada campo tiene un significado diferente. Aún así, existen muchas maneras de almacenar una dirección en el modelo relacional. En una forma mínima, la dirección se puede guardar como una gran cadena de caracteres; pero entonces se perdería la estructura. El usuario se tendría que preocupar entonces acerca de los diferentes campos y atributos dentro de todo el campo. La calificación o manejo de los diferentes campos de una dirección se vuelve muy difícil o imposible.

Otra posibilidad sería la captura los diferentes campos de la dirección usando distintos atributos, tales como campos para NumeroDirección, NombreDirección, CiudadDirección, EstadoDirección y CPDirección. En esta opción se pierde la agregación lógica de los campos de dirección. La definición de la tabla se ve extraña y el usuario debe hacer la asociación de los diferentes campos en el mismo objeto lógico.

Una tercera alternativa sería almacenar las direcciones en tablas separadas y realizar reuniones para recuperar los diferentes campos de la dirección. Aunque esta opción evita algunas de las inconveniencias de las otras dos alternativas, necesita realizar muchas reuniones para recuperar los campos de la dirección de una persona. Aún más, la tabla de direcciones no es una agrupción interesante, debido a que, en la mayoría de los casos, los usuarios recuperan las direcciones a través del registro de una persona.

La representación más natural para una dirección es permitir atributos valuados por tuplas. Con esta alternativa, la dirección sería una tupla y existiría como una entidad lógica sencilla, en vez de estar diseminada en múltiples campos o tablas. El usuario podría accesar y actualizar cada campo por separado.

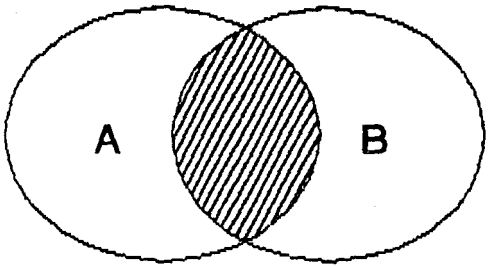


Figura 3.5(b). Interseccion entre conjuntos.

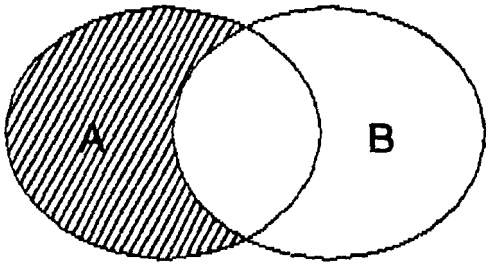


Figura 3.5(c). Diferencia entre conjuntos.

3.2.4.2. Modelos Semánticos.

El objetivo de los modelos semánticos es el modelaje de abstracciones estructurales, no se incorporan abstracciones de comportamiento (o tipos abstractos de datos). En este sentido, este modelo es similar a la representación de conocimiento de la Inteligencia Artificial; los modelos semánticos intentan capturar la semántica de los objetos (entidades) y sus relaciones dentro de un espacio de objetos subyacente. Estos modelos son usados principalmente como herramientas de diseño de bases de datos relacionales o de red.

Los modelos semánticos usan una representación de objetos basado en los conceptos de nodo y enlace, donde cada nodo es un tipo de entidad; este concepto de nodo es similar al concepto de tipos en los lenguajes de programación. Un tipo de entidad representa un conjunto de objetos, todos compartiendo los mismos atributos. Un atributo es una función que se puede aplicar a una entidad o tipo de entidad; el nombre de un tipo de entidad también identifica la extensión (conjunto de todas las instancias) de ese tipo de entidad. Los tipos de entidad son análogos a las clases y las entidades a las instancias; los atributos son similares a las variables de las instancias.

Los modelos semánticos capturan también la herencia e incorporan varios tipos de atributos (o tipos base en los lenguajes de programación).

3.3. SISTEMAS DE ADMINISTRACION DE BASES DE DATOS ORIENTADOS A OBJETOS (OODBMS).

3.3.1. Influencias de los Modelos de Datos en el Desarrollo de las Bases de Datos.

Todos los DBMS, para su diseño, están basados en algún concepto de modelaje de datos, aunque esto a veces no es cierto: la mayoría de las veces, las llamadas DBMS relacionales no incorporan totalmente los conceptos del modelo relacional. Igual sucede con los DBMS orientados a objetos; a veces no incorporan totalmente algún modelo de datos; a veces incorporan varios modelos; a veces introducen diversas estrategias de desarrollo que utilizan en diferentes grados los diferentes modelos de datos. Lo único seguro es que no existen estándares establecidos para considerar un DBMS como "orientado a objetos"; el resultado a esta falta de estandarización es el siguiente: nadie sabe realmente cómo debe ser un OODBMS, por lo que aún los DBMS que soportan parcialmente los conceptos orientados a objetos pueden ser reconocidos como <<sistemas orientados a objetos para la administración de bases de datos>>.

3.3.2. Características.

Debido a que no existen estándares acerca de los OODBMS, es necesario usar el término <<DBMS orientada a objetos>> en su sentido más genérico, combinando características orientadas a objetos con capacidades "tradicionales" de las bases de datos; de los conceptos orientados a objetos se puede concluir:

Orientación a Objetos =
Tipos Abstractos de Datos + Herencia + Identidad de Objetos

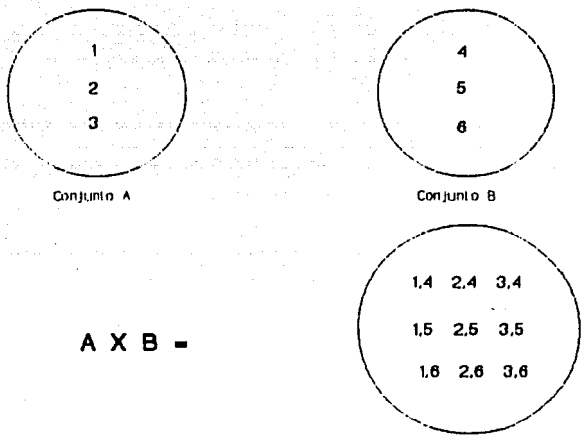


Figura 3.5 (d). Producto Cartesiano.
Las figuras 3.5 (a) hasta 3.5 (d) son llamadas operaciones entre conjuntos.

Adicionalmente a estos elementos, las bases de datos orientadas a objetos deben soportar (idealmente) los siguientes conceptos relacionados con las bases de datos:

1. Persistencia.
2. Transacciones.
3. Control de concurrencia.
4. Recuperación.
5. Consultas.
6. Versionamiento.
7. Integridad.
8. Seguridad.
9. Características de desempeño.

De esta manera podemos definir una base de datos orientada a objetos ideal como:
Base de Datos Orientada a Objetos =
Orientación a Objetos + Capacidades de Bases de Datos.

Persistencia. Esta característica se refiere a que un objeto debe existir fuera del contexto de una transacción llevada a cabo, es decir, que debe guardarse de forma independiente al sistema. De esta forma, cuando el administrador de bases de datos termina su ejecución, los objetos manipulados no se pierden.

Transacciones. Una transacción es un programa que se ejecuta enteramente o no se ejecuta; a esto se le llama atomicidad, las transacciones son atómicas. De forma ejemplificada: los objetos o entidades o los datos que el sistema de base de datos puede accesar, existen independientemente de éste; para que el sistema pueda accesar estos datos tiene que realizar la acción de leerlos (o cargarlos en memoria, o abrir un archivo; son aspectos técnicos no relevantes). A esta acción se le llama transacción. En forma genérica, una transacción es un programa (o una parte del sistema) que permite que el sistema accese sus datos, ya sea para leerlos, para alterarlos, para borrarlos, etc.

Control de concurrencia. En el medio ambiente típico de ejecución de un DBMS, las transacciones corren en forma concurrente. Esto quiere decir que puede haber distintas transacciones activas al mismo tiempo; aún en el campo de las computadoras personales esto puede aplicarse: existen sistemas operativos multitarea, donde esto se puede hacer, y en las computadoras interconectadas (conectadas en red), una base de datos puede tener varias transacciones activas al mismo tiempo. Por otro lado, en los sistemas operativos que no ofrecen multitarea, la concurrencia se refiere a que el sistema de administración puede tener varias tablas activas a la vez, aunque solamente puede trabajar con una a un tiempo.

Dado que las transacciones permiten accesar los datos persistentes de una base de datos, el sistema debe garantizar la consistencia de los datos persistentes y los resultados de una transacción. En los sistemas multitarea y en las redes de computadoras, esto se traduce a que el sistema es responsable de saber a que tabla o archivo pertenece alguna entidad y de darle seguimiento a que usuario está realizando transacciones con cual entidad perteneciente a cual base de datos; en

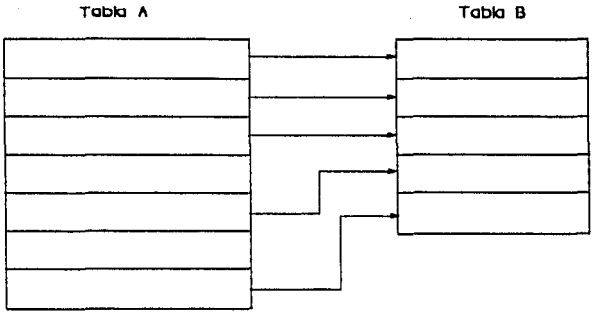


Figura 3.5(e). Operación Selección. Se seleccionan ciertas tuplas siempre y cuando cumplan alguna condición dada. La tabla B es una selección de A.

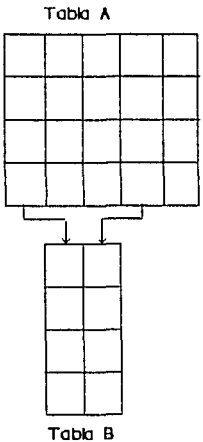


Figura 3.5(f). Operación Proyección. Se proyectan ciertas columnas seleccionadas por el usuario. La tabla B es un proyección de A.

los sistemas no multitarea esto se traduce a que el sistema debe saber que datos pertenecen a cual tabla o base de datos, si hay varias activas.

Recuperación. Dado que las transacciones son atómicas, el DBMS debe garantizar que los resultados parciales o las actualizaciones parciales de transacciones fallidas no se propagen a la base de datos persistente.

Consultas. Las transacciones son acciones que permiten accesar los datos. Las consultas son acciones que permiten especificar cuales datos se van a accesar. Técnicamente, las consultas son programas que ejecutan diferentes transacciones, es decir, con una consulta se pueden recuperar ciertos datos, operar sobre ellos, o borrarlos selectivamente.

Versionamiento. Este término es una adaptación libre al término original en inglés "versioning" y se refiere a la capacidad de accesar estados previos o alternos de objetos con un estado actual. Esta capacidad permite recuperar las diferentes versiones de un objeto; obviamente, esto es aplicable cuando un objeto ha sufrido varios cambios o actualizaciones. Ciertamente en todas las aplicaciones de las bases de datos, los datos se actualizan cada cierto tiempo, por lo que ciertamente esta característica es ampliamente aplicable.

Integridad. La consistencia de una base de datos es el nivel de coherencia que tienen los datos o entidades de esa base de datos. Esta consistencia se puede expresar a través de predicados o condiciones del estado actual de la base de datos; los predicados también se pueden aplicar a objetos o a valores de atributo en la base de datos. Los predicados que capturan la consistencia de una base de datos son llamados condiciones de integridad. En forma general, existe un cierto numero de condiciones de integridad que se deben reforzar en un estado de la base de datos para garantizar su consistencia. Los siguientes son algunos ejemplos:

1. La edad de una persona no puede ser un numero negativo.
2. El balance de alguna cuenta debe ser menor o igual a la suma de los depósitos.
3. Si un empleado trabaja en algun departamento en particular, entonces debe existir un registro en la base de datos para ese departamento.
4. El numero de seguro social para cada empleado debe ser único para el conjunto de todos los empleados.
5. Una persona debe tener un nombre; el atributo Nombre no puede estar vacio o ser nulo.

Seguridad. Los DBMS deben facilitar al usuario el uso de primitivas de seguridad para colocar diferentes tipos de acceso a las bases de datos o archivos de información. Esto es especialmente importante en ambientes operativos multiusuario, donde cada usuario debe tener ciertos privilegios de acceso a los datos; el operador del sistema (visto por el programa como otro usuario) debe poder utilizar ciertas instrucciones para colocar ciertas restricciones a los otros usuarios. Esto también es aplicable a sistemas para un solo usuario: dado que varios usuarios pueden usar el sistema de computación, el programa de

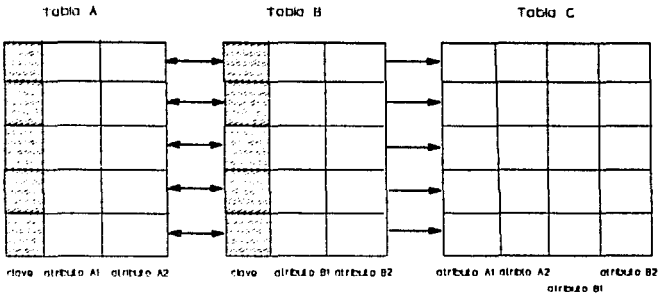


Figura 3.5(g). Operación Reunión, las eneadas de 2 tablas se reunen en otra tabla según alguna eneada común (en éste caso clave). La tabla C es la reunión de A y B (C = A reunión B).

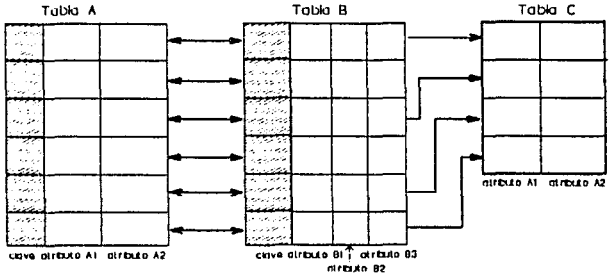


Figura 3.5(h). Operación división. La división de la tabla B en A es la tabla C que tiene ciertas eneadas de A de acuerdo a una condición que se aplica a las eneadas en B. En este ejemplo, C = B división A.

Las figuras 3.5 (e) hasta 3.5 (h) son conocidas como operaciones relacionales.

base de datos debe proporcionar facilidades para permitir un acceso restringido a los datos de las bases de datos.

Caracteristicas de desempeño. El desempeño de los sistemas de administración de bases de datos, como el de muchas otras aplicaciones, es extremadamente importante. Si dos sistemas proporcionan una funcionalidad similar, la elección de uno de los dos se basa frecuentemente en su desempeño o en pruebas estandarizadas. Estas pruebas son programas que se usan para medir el desempeño de algunas propiedades criticas del sistema: el tiempo de respuesta para algunas consultas complejas; qué tan rápido puede ejecutar actualizaciones; cuantos usuarios pueden accesar la base de datos concurrentemente; etc. [EHU90].

3.3.3. Bases de Datos Relacionales y Bases de Datos Orientadas a Objetos.

Los DBMS relacionales, se han quedado estancados en la forma en que manejan los datos y, lo mas importante, la forma de los datos que manejan. Estos dos puntos se traducen en la dificultad de traducir información producida por el mundo real a estructuras de información que se puedan almacenar en la base de datos; pero lo más trascendente radica en el hecho de que se han desarrollado muchas aplicaciones independientes a los DBMS: sistemas de ayuda de diseño, sistemas contables, de manejo de documento, organizadores gerenciales, etc., estos sistemas tienen una caracteristica en comun: todos utilizan datos; muchas veces los mismos datos que se utilizan en un programa de contabilidad son los datos que se quieren graficar, quizá se requieren estos datos para organizar una planeación administrativa, y a veces es necesario tener estos datos disponibles en una base de datos para poder hacer distintas organizaciones estructurales y generar reportes en serie. Todo esto apunta a la creación de objetos de datos que puedan ser reconocidos por distintos sistemas, entre ellos DBMS.

Además del problema de falta de conectividad con otros sistemas, se han identificado otros problemas de los DBMS relacionales, problemas que se intentan resolver a través de los DBMS orientados a objetos:

1. La brecha semántica. Este problema se refiere a las diferentes representaciones entre el dominio de una aplicación y la base de datos persistente almacenada en el sistema de computación. En los OODBMS, el mundo real se modela tan fielmente como es posible y los enlaces y relaciones entre entidades de la compleja realidad se representan y manipulan directamente, reduciendo asi esta "brecha semántica" entre las distintas representaciones.

2. Desajuste de impedancias. Los OODBMS también reducen el "desajuste de impedancias" entre el lenguaje de programación y el sistema de administración de bases de datos. En aplicaciones complejas, se recuperan datos de una base de datos usando lenguajes de consulta de bases de datos tales como SQL y entonces se manipulan los datos a través de rutinas escritas en lenguajes de programación más convencionales como C o Pascal. Los lenguajes de programación son procedurales;

los lenguajes de consultas de bases de datos son de un nivel más alto y por tanto más declarativos. Por tanto, las aplicaciones que involucran el uso de ambos tipos de lenguajes causan un desajuste de paradigmas. Aun más, los tipos de datos declarados en los diferentes lenguajes (SQL y C, por ejemplo) no son los mismos y tienen que ser mapeados entre los lenguajes usados. La razón de accesar los datos a través de un lenguaje de base de datos y entonces procesarlos a través de un lenguaje convencional es porque generalmente los lenguajes de base de datos están limitados a declarar consultas. Típicamente, los lenguajes de bases de datos proporcionan poco soporte para cálculos complejos, estos deben hacerse en un lenguaje de programación general. Los OODBMS son más completos en el sentido de que típicamente proporcionan mayor expresividad.
El lenguaje de manipulación de datos (DML) de un OODBMS debe proporcionar construcciones de programación de propósito general:

a. Estructuras de control (IF-THEN-ELSE, ciclos WHILE, sentencias CASE, etc.)
b. Sentencias de declaración y manipulación de variables.
c. Una gran colección de tipos de datos o clases.
d. Construcciones para definir métodos, funciones, tipos abstractos de datos (clases), etc.
En realidad, algunas bases de datos convencionales incorporan algunas construcciones generales de programación a través de lenguajes de cuarta generación o dialectos de SQL; por ejemplo, tanto TransactSQL de Sybase y dBASE/SQL de Ashton-Tate incluyen estructuras de control, variables, procedimientos y otras construcciones de programación.

3. Requerimientos de aplicaciones de ingeniería y automatización de oficinas. Los mercados principales de los OODBMS han sido las aplicaciones para ingeniería: Diseño Asistido por Computadora (Computer-Aided Design, CAD), Manufactura Asistida por Computadora (Computer-Aided Manufacture, CAM), Ingeniería de Software Asistida por Computadora (Computer-Aided Software Engineering, CASE) junto con aplicaciones de automtización de oficinas. Estas aplicaciones tienen características que son diferentes de aquellas de las aplicaciones para negocios más tradicionales o de contabilidad; en las aplicaciones para ingeniería y para automatización de oficina, la cantidad de información almacenada por la base de datos es mucho más grande y las correlaciones de la información son más complejas. El tamaño y la complejidad de los datos hacen casi imposible agrupar todas las permutaciones posibles de los datos relacionados; por esto, se hacen necesarias una cantidad considerable de maniobras para poder accesar y actualizar objetos en la base de datos. En las aplicaciones típicas de diseño, cada parte de un sistema sufrirá muchos refinamientos y tendrá múltiples versiones o alternativas. Muchas de las características de los OODBMS son intentos de satisfacer estos requerimientos [KH090].

3.3.4. Estrategias de Desarrollo.

Existen por lo menos seis enfoques para incorporar caracteristicas orientadas a objetos a las bases de datos:

1. Nuevo enfoque de modelo/lenguaje de datos para bases de datos. El enfoque más agresivo es el desarrollo de un lenguaje de base de datos y un DBMS enteramente nuevo con capacidades orientadas a objetos. La mayoria de los proyectos de investigación en OODBMS utilizan este enfoque.

2. Extensión de un lenguaje de base de datos existente con capacidades orientadas a objetos. A muchos lenguajes de programación generales se les han incorporado construcciones orientadas a objetos; por ejemplo, C++, Flavors (una extensión de LISP), Object Pascal, entre otros. Es probable que se siga un enfoque similar con los lenguajes para bases de datos. Debido a que SQL es el estándar y el lenguaje de base de datos más popular, la solución más razonable es la extensión de este lenguaje con construcciones orientadas a objetos, reflejando las capacidades orientadas a objetos del DBMS subyacente. Muchos vendedores de DBMS relacionales siguen este enfoque, conforme evolucionan sus productos; ha habido muchos intentos para incorporar la herencia, composición de funciones o entidades anidadas y hasta encapsulación en un marco de trabajo SQL.

3. Extensión de un lenguaje orientado a objetos con capacidades para bases de datos. Otro enfoque es la incorporación de capacidades para bases de datos en algún lenguaje orientado a objetos existente; de esta forma, las características de la orientación a objetos (tipos abstractos de datos, herencia, identidad de objetos) ya estarian soportados en el lenguaje de programación, las extensiones incorporarian caracteristicas de bases de datos (consultas, soporte de transacciones, persistencia, etc.).

4. Proporcionar bibliotecas extensibles orientadas a objetos para el manejo de DBMS. Asi como se pueden incorporar capacidades de manejo de bases de datos en algún lenguaje orientado a objetos, se pueden introducir bibliotecas independientes accesibles desde cualquier lenguaje orientado a objetos. De esta manera, no es necesaria la modificación de algún compilador. Estas bibliotecas pueden incluir clases para agregados (conjuntos, listas, arreglos) y tipos, métodos para empezar/realizar/abortar transacciones, manejo de excepciones y agrupamiento de objetos.

5. Incrustamiento de construcciones orientadas a objetos de un lenguaje de base de datos en un lenguaje huesped (convencional). Los lenguajes de bases de datos se pueden

incrustar en lenguajes de programación huéspedes, por ejemplo: se pueden usar sentencias SQL en PL/1, C, FORTRAN, COBOL y Ada (cuando esto sucede, se dice que el SQL está incrustado en un lenguaje huesped). Los tipos de SQL (esto es, relaciones y enedadas en relaciones) son muy diferentes a los sistemas de definición de tipo en el lenguaje huesped. Algunos OODBMS han tomado este enfoque con un lenguaje huesped y un lenguaje de base de datos orientado a objetos.

6. Productos para una aplicacion especifica con un sistema orientado a objetos subyacente. Otro enfoque interesante es el desarrollo de cierto numero de herramientas y medios ambientes especificos a una aplicación/dominio que usan tecnología de base de datos orientada a objetos o proporcionan visibilidad de una base de datos orientada a objetos al dominio de una aplicación. La intención de las soluciones especificas a un dominio o aplicación no es proporcionar un medio ambiente de propósito general; solamente se hace visibles al usuario construcciones útiles o específicas a la aplicación, posiblemente con algunas caracteristicas orientadas a objetos. [kH090].

3.3.5. Ventajas.

Hasta este punto, se han presentado las características que proponen los diversos modelos de datos, a partir de esta información se puede entrever diversas ventajas (y desventajas) entre ellos. En forma concreta, las ventajas potenciales de los OODBMS son las siguientes:

- Representaciones específicas de tipos altamente eficientes.
- Aislamiento (encapsulación) de la aplicación.
- Consistencia garantizada de las operaciones realizadas por diferentes aplicaciones en objetos de un tipo dado.
- Reducción notable del costo de desarrollo de nuevas aplicaciones debido a la herencia de código de tipos predefinidos de objetos [AfW90].

3.3.6. Ejemplos de Sistemas de Administración de Bases de Datos Orientadas a Objetos.

3.3.6.1. Prototipos de Investigación.

Hay muchos prototipos de investigación de bases de datos orientadas a objetos, algunos de estos han hecho contribuciones significativas al campo. He aquí algunos prototipos de investigacion.

FAD. Uno de los primeros prototipos estaba basado en el lenguaje de datos orientados a objetos FAD (Bancilhon et al., 1987). FAD son las siglas de Franco-Armenian Database language (lenguaje de Base de Datos Franco-Armenio). El primer prototipo se construyó en el consorcio MCC (Microelectronic and Computer Corporation) y se completó en 1987. El lenguaje FAD estaba basado en el cálculo lambda y proporcionaba un claro soporte para identidad de objetos; FAD fue implantado en un ambiente distribuido y también se podia usar almacenamiento e indexación para objetos complejos.

ORION. MCC desarrolló también otro prototipo de investigación de base de datos orientada a objetos llamada ORION (Banarjee et al., 1987). El proyecto ORION proporcionaba algunas características interesantes para OODBMS, tales como la evolución de esquemas y el bloqueo de objetos complejos (Kim et al., 1987).

IRIS. IRIS (Fishman et al., 1987,1989; Beach, 1988) es un prototipo de un DBMS de siguiente generación desarrollado en los laboratorios Hewlett-Packard. El modelo de datos Iris soporta la herencia y los tipos abstractos de datos y está basado en modelos funcionales; también proporciona facilidades para compartir referencialmente los objetos definidos. Una de las interfaces de Iris es una extensión orientada a objetos de SQL llamada Object SQL. Iris también incorpora un mecanismo de control de versiones.

Galileo. Este es un lenguaje fuertemente tipeado (Albano et al., 1985) que fue desarrollado en la Universidad de Pisa, en Italia. Galileo incorpora herencia de tipos y soporte para objetos complejos. Esta es una de las pocas bases de datos que intentan distingir entre las nociones de tipo y clase.

Otros Sistemas. Además de estos prototipos de investigación, existen muchos otros, tanto en la universidades como en la industria. Entre estos está el modelo de datos PROBE, desarrollado por la Computer Corporation of America (Manola y Dayal, 1986); el sistema de base de datos ENCORE desarrollado en la Universidad Brown (Zdonik y Wegner, 1986); el sistema de manejo de base de datos extensible EXODUS de la Universidad de Wisconsin, Madison (Carey et al., 1990); el medio ambiente para especificar e implantar sistemas de bases de datos ADAPTABLE (Stemple et al., 1990); el sistema de base de datos POSTGRES de la Universidad de California, Berkeley (Stonebraker y Rowe, 1986; Stonebraker et al., 1990); el sistema de base de datos orientada a objetos O2 del proyecto Altair (Deux et al., 1990), entre otras [KHO90].

3.3.6.2. Sistemas Comerciales.

Los primeros OODBMS comerciales comenzaron a aparecer a mediados de la década de los 80's. Los creadores de estos primeros sistemas realizaron una labor pionera definiendo los componentes de los OODBMS y probando el mercado para las aplicaciones de esta nueva tecnología. A continuación se presentan algunos de los sistemas disponibles actualmente:

G-Base. Uno de los primeros productos de OODBMS fue G-Base de Graphael, que fue presentado en 1987. G-Base es un sistema basado en LISP; está escrito en LISP y las entidades y los métodos son definidos usando sintaxis de LISP. G-Base soporta los tipos abstractos de datos y herencia múltiple de clases; el medio ambiente incluye también un manejador interactivo de objetos gráficos, un generador de reportes interactivo y un lenguaje de consultas interactivo. El modelo de base de datos está compuesto de entidades y relaciones. Los tipos abstractos de datos se incorporan permitiendo que los usuarios definan los métodos para las entidades; G-Base también puede establecer una interfaz con DBMS relacionales. Otra herramienta importante de G-Base es G-Logis, la cual integra inferencia (por medio de una extensión de

Prolog) en el medio ambiente G-Base, por tanto, G-Base proporciona capacidades elegantes para modelar, manipular y consultar muchos datos.

GemStone. El sistema GemStone (Servio Logic, 1987, 1989) para administración de base de datos orientada a objetos es uno de los primeros DBMS disponibles y también uno de los más influyentes en los DBMS de siguiente generación, y fue desarrollado por la Servio Logic Corporation de Beaverton, Oregon. El sistema GemStone es significativo en muchos sentidos: fue uno de los primeros OODBMS que proporcionó un fuerte soporte para los conceptos de identidad de objetos, herencia y encapsulación, junto a muchas capacidades de bases de datos como persistencia, transacciones y consultas ad-hoc. El lenguaje de definición y manipulación de datos es llamado OPAL, y está basado en el lenguaje Smalltalk; por eso, la sintaxis de OPAL es muy parecida a la de Smalltalk. Empero, OPAL incorpora muchas construcciones y operaciones específicas para el manejo de bases de datos.

Ontos. El OODBMS Ontos (Ontologic, 1988, 1989) fue desarrollado por Ontologic, Burlington, Massachusets. Ontos es un sistema basado en C++ y ofrece una biblioteca cliente para C++ que sirve de interfaz entre el programa de aplicación y la base de datos persistente. Ontos tiene un predecesor llamado Vbase, también de Ontologic, que proporcionaba una extensión de C para definir métodos y un lenguaje de definición de tipos para crear tipos de entidades y el esquema; por eso, muchos de los conceptos de Ontos vienen de Vbase. La biblioteca C++ de Ontos incluye clases para definir colecciones de objetos tales como conjuntos y otros tipos agregados. El modelo Ontos soporta la encapsulación, herencia y la habilidad para contruir objetos complejos; en términos de capacidades de base de datos, Ontos soporta transacciones concurrentes y anidadas de objetos persistentes.

Statice. Este es un OODBMS desarrollado por Symbolics (Symbolics, 1988). Soporta herencia en tipos de entidad y de métodos que pueden ser asociados con tipos de entidad; también se soportan atributos valuados en entidades y en conjuntos. La sintaxis del lenguaje de definición y manipulación de datos de Statice es similar a la de LISP; en términos de capacidades de bases de datos, se soportan transacciones concurrentes, recuperación, acceso asociativo y consultas.

SIM. SIM (Jagannathan et al., 1988, UNISYS, 1987) son las siglas de Semantic Information Manager (Administrador Semántico de Información) y es un DBMS comercial basado en el modelo de datos semántico, de UNISYS Corporation. El sistema SIM es el nucleo para el InfoExec Environment (Medio ambiente InfoExec) de UNISYS; otros productos en InfoExec incluyen un sistema de diccionario de datos, facilidades para hacer consultas interactivas y componentes de interfaz de lenguaje. El modelo de datos en SIM está basado en el Modelo Semántico de Datos de Hammer y McLeod (1981); de esta manera, se pueden definir tipos de entidad (clases) que se pueden heredar, y el modelado de objetos complejos y de relaciones complejas entre entidades se puede realizar muy fácilmente. Las funciones de atributo pueden estar valuadas en forma sencilla o multiple. SIM también permite la especificación de varias formas de condiciones de integridad.

Otros sistemas. Además de los productos "puros" de OODBMS, existen algunos sistemas relacionales que están evolucionando incrementalmente hacia la orientación a objetos. Las características que están incorporando estos sistemas son:

1. Funciones definidas por el usuario.
2. Tipos abstractos de datos definidos por el usuario.
3. Campos multimedia muy grandes.
4. Extensiones de funcionalidad y estructura (herencia de tablas y de tuplas [registros], identidad de entidades y tipos abstractos de datos para tablas y tuplas).

CAPITULO 4

Analisis Comparativo y Diseño de un Sistema de Administración de Bases de Datos Orientada a Objetos.

4.1. ANALISIS COMPARATIVO DE UNA BASE DE DATOS USANDO EL MODELO ORIENTADO A PROCEDIMIENTOS Y ORIENTADO A OBJETOS.

4.1.1. Analisis de Requerimientos.

El primer paso a dar cuando se va a desarrollar algun sistema es definir el problema. Saber qué es lo que se va a resolver, definir la problematica en un enunciado, para poder proponer una solución. Dentro del ámbito de trabajo de esta tesis, se tienen ya varios delimitadores del campo de acción: es necesario que el problema propuesto sea lo suficientemente amplio para poder realizar un desarrollo orientado a procedimientos y orientado a objetos; esto es, el problema no debe ser ad hoc para un desarrollo orientado a objetos, o para la aplicación de este paradigma. El problema debe ser lo suficientemente general para derivar de el diversos programas de aplicación, es decir, el problema debe plantear un esquema en el sentido más general del término para poder mostrar conceptos orientados a objetos sin tener que comprimirlos en una sola aplicación y también para poder tener a la vista el objetivo de mostrar un diseño orientado a objetos para una base de datos, sin tener que perderse en detalles de diseño de pantallas, o en captura de datos. Empero, el problema deberá ser aplicable a la realidad para poder manejar datos reales y evitar las idealizaciones que podrían comprometer al diseño orientado a procedimientos o el diseño orientado a objetos.

La aplicación que se va a desarrollar trata acerca del almacenamiento de información escolar. El enunciado del problema es el siguiente:

Se necesita guardar información acerca del alumnado en la escuela, tanto para alumnos inscritos en el ciclo escolar como para alumnos que ya tienen el 100% de los créditos de su carrera (pasantes), y también información acerca del profesorado en la escuela; el sistema servirá para guardar y recuperar información. La recuperación de información debe producir reportes (tanto en pantalla como en impresora) que contengan información en una forma que sea útil. Los reportes deben proporcionar los siguientes informes:

- Los datos generales de cada persona, sea estudiante inscrito, pasante o profesor. Estos datos deben incluir el nombre, dirección, teléfono, sexo y para alumnos y pasantes, el numero de cuenta.

- Para alumnos inscritos, el sistema debe recuperar el porcentaje de créditos cursados.

- El sistema debera tener la capacidad de decidir si un alumno o pasante es (o fue) un buen estudiante, de acuerdo al siguiente criterio:
  Si tiene (tuvo) un promedio mayor o igual a 8.5;
  si participo en alguna actividad extraescolar reconocida.

- Deben identificarse los alumnos que, según los estatutos

universitarios, ya no deberían estar inscritos. Los estatutos universitarios marcan que el periodo legal es igual al número de semestres marcados por el plan de estudios actual de la licenciatura mas el 50% adicional de tiempo.

- El sistema deberá reconocer cuales alumnos son regulares, de acuerdo al siguiente criterio:
  Un alumno es regular si tiene promedio mayor que 6 y
  si está dentro de su periodo legal de estudios.

- Si el alumno es regular, se necesita saber cuantas materias lleva, el semestre a que está inscrito actualmente y los créditos que se proyectan para (que aprobará) este semestre.

- Se deben reconocer los pasantes que tengan registrada la realización de su servicio social y también se deberán reportar los siguientes datos: el lugar donde se realiza o realizó el servicio, en qué fecha va a empezar, el proyecto o actividad realizada o a realizar y el nombre del responsable o asesor del servicio. El sistema deberá informar si ya terminó el servicio, reportando la fecha de trámite de la carta de terminación.

- De igual manera se debe reportar la información acerca del registro de la tesis; la fecha de registro (y la fecha de vencimiento de ese registro) y el título de la tesis.

- También se necesita la información acerca de los pasantes que ya tengan en trámite la petición de la fecha de exámen profesional, se necesita saber la fecha asignada y los nombres de los sinodales.

- En cuanto a los profesores, el sistema debe recuperar el sueldo nominal.

- Se necesita que el sistema pueda reportar los datos de los profesores que tengan asignada una cierta materia, que tengan una cierta profesión o que tenga cierto conocimiento en otras areas del conocimiento (por ejemplo, un profesor cuya profesión sea ingeniería civil, pero que esté capacitado en el area de la computación).

- Asimismo, se necesita tener la información acerca del número de horas asignadas y cuantas materias tiene asignadas.

4.1.2. Objetivos del Sistema.

El objetivo principal del sistema es producir información que tenga utilidad; quedó establecido que esta información versará sobre la escuela, reportando los informes pedidos.

Otro objetivo es que éstos reportes deben ser producidos como consultas y en forma impresa; la producción de consultas se refiere a que el usuario debe tener la posibilidad de solamente consultar la información pedida. Físicamente, los informes deben aparecer en pantalla para hacer consultas. Los reportes en forma impresa se refieren a que el usuario debe tener la posibilidad de tener la información pedida en papel impreso y en forma permanente.

Otra facilidad que deberá otorgar el sistema es la adición o borrado de información, así como la posibilidad de hacer cambios o actualizaciones a la información existente.

Es decir, lo que debe hacer el sistema es tomar la información disponible o necesaria de una base de datos persistente y seleccionarla o procesarla de forma que se produzcan los informes pedidos. Hasta este punto no es necesario comprometerse con algún método de desarrollo, orientado a procedimientos u orientado a objetos, las diferencias estriban en la selección de una estrategia de trabajo o funcionamiento del sistema.

4.1.3. Diagrama de Relaciones de Entidades.

Dentro del paradigma orientado a objetos, se habla de que el administrador de una base de datos sólo debe controlar las comunicaciones entre entidades de información, todo el trabajo (comparaciones, selección de información, construcción de reportes) en realidad es realizado por estas entidades.

Según los requerimientos pedidos, son necesarias tres entidades: una que controle toda la información referente a los estudiantes inscritos, una que controle la información acerca de los pasantes y una entidad que controle toda la información sobre los profesores. No es necesario que estas entidades tengan comunicación entre sí; lo único necesario es otra entidad que tenga comunicación directa con las entidades representantes de alumnos, pasantes y profesores para que pueda pasarles los comandos del usuario, es decir, es necesaria una entidad de control que pueda comunicarles a las otras entidades las condiciones de búsqueda dadas por el usuario (por ejemplo, el usuario puede dar una condición <<recuperar los datos de un estudiante cuyo nombre sea Juan Pérez>>, la entidad de control lanza este mensaje a la entidad de control de alumnos y ésta última entidad es la que se va a encargar de buscar y recuperar la información pedida), esto también implica que esta entidad de control tendrá interacción directa con el usuario. Al hecho de que la entidad de control pueda comunicar alguna condición de búsqueda se le llamará "la entidad de control lanza una convocatoria" y a la información o reportes generados por las otras entidades se denominará "las entidades dan información de respuesta" o simplemente que "las entidades responden a la convocatoria".

La figura 4.1 ilustra este concepto de comunicaciones entre las entidades primarias (u objetos que se han identificado hasta este punto).

4.1.4. Diagrama Orientado a Procedimientos.

Siguiendo el paradigma orientado a procedimientos, el sistema se tiene que descomponer en módulos o subprogramas que realicen algún proceso simple. Según la filosofía del análisis estructurado, es conveniente empezar por los módulos principales e ir distribuyendo el trabajo necesario. Empezando con un módulo principal (al que se le

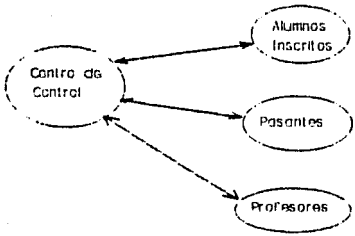


Figura 4.1 Diagrama de relaciones entre entidades.

Módulo de control

- Seleccionar el procedimiento para recuperar la información pedida, pasarle los parametros adecuados a ese procedimiento

Selección
- Procedimiento A
- Procedimiento B
- Procedimiento C

etc.

Para cada procedimiento:
- Accesar la tabla correspondiente
- Llamar a un subprocedimiento específico según los parametros.

- procedimiento a
- procedimiento b
- procedimiento c

etc.

Figura 4.2. Diagrama orientado a procedimientos

llamará "módulo de control") que controlará los comandos del usuario y pasará el control a otros subprogramas específicos a los parámetros del comando (esto es, si el comando hace referencia a los datos de los profesores, el módulo de control llamará a un subprograma que accese y procese datos de la tabla de profesores). A su vez estos subprogramas deberán procesar los parametros dados y llamar a subprogramas subordinados que recuperen la información pedida, segun el parámetro (por ejemplo, si el comando dado por el usuario fue <<recuperar los datos de un estudiante cuyo nombre sea Juan Pérez>>, el módulo de control deberá invocar un subprograma que pueda accesar a la tabla de estudiantes y pasarle los parámetros <<nombre = Juan Pérez>>; a su vez, este subprograma deberá invocar a otro subprograma que pueda accesar el campo "nombre" de la tabla accesada y buscar la ensada con el nombre "Juan Pérez").

Esta forma de funcionamiento está ilustrada en la figura 4.2 (esta figura no sigue los lineamientos estándares para la diagramación estructurada, sólo se ofrece como una forma de ilustración para los conceptos anteriores).

4.2. DISEÑO DE UN SISTEMA ORIENTADO A OBJETOS PARA LA ADMINISTRACION DE BASES DE DATOS.

4.2.1. Técnicas de Diseño.

El diseño de cualquier sistema usando cualquier metodología, consiste en el refinamiento de los módulos, partes o entidades que conforman el sistema. En este sentido, el diseño del proyecto actual usando un paradigma orientado a procedimientos consiste en la definición y refinamiento de todos los módulos y subprogramas necesarios identificados en el análisis. Es decir, el diseño del DBMS planteado en el análisis anterior, usando técnicas orientadas a procedimientos, consiste en precisar los módulos necesarios y las comunicaciones (interfaces) entre ellos; el diseño usando técnicas orientadas a objetos consiste en definir y refinar las entidades identificadas como necesarias en el análisis anterior. En realidad, las dos metodologías de diseño no son muy distintas entre sí; la única diferencia sustancial entre ambas está en la forma en cómo se definió el funcionamiento o la estrategia de funcionamiento del sistema, estrategias que se pueden apreciar en las figuras 4.1 y 4.2.

4.2.2. Diseño de Transacciones.

La primera transacción que se va a incluir será la edición de datos, es decir, agregar datos, cambiarlos y borrarlos. Desde el punto de vista orientado a objetos, esta transacción se traduce como el acceso (o la creación de una nueva instancia [objeto] de la entidad definida y manipulación de la información ahí contenida. Desde un punto de vista orientado a procedimientos, esta transacción significa accesar la base de datos persistente y tomar (o agregar) un nuevo registro; en este registro, modificar (o agregar) información a los campos del registro (modificar columnas de la tupla) y dejarlo en estado permanente como integrante de la base de datos.

Esta forma de manipular información (una OODBMS toma objetos o instancias de una entidad y modifica la información contenida y una DBMS "convencional" accesa una tupla de una tabla y manipula la información de sus columnas) conforma la estrategia cimiento donde se apoyará el funcionamiento del sistema.

En los diagramas 4.1 y 4.2 se mostró la filosofía de funcionamiento de dos DBMS según los paradigmas orientados a objetos y orientados a procedimientos. El diseño general orientado a objetos solamente expande los conceptos del diagrama de relaciones de entidades; en este diagrama se muestra que las entidades identificadas (Inscritos, Pasantes y Profesores) tienen una participación activa dentro del funcionamiento del sistema, esto es, que parte del funcionamiento del sistema dependerá de las acciones de las entidades que forman parte de él. Las entidades no son sólo estructuras de datos que esperen pasivamente a que algún módulo actúe sobre él. La participación activa de las entidades quiere decir que las transacciones requeridas se realizan en las mismas entidades. En el apartado 3.1.3 se mencionó en forma general la manera en que se iban a comunicar las entidades integrantes del sistema: la entidad de control lanza una convocatoria y las entidades de datos responden a esta convocatoria. La entidad de control lanza la convocatoria de acuerdo a las órdenes del usuario; esto es, la entidad de control debe controlar módulos que le presenten al usuario opciones para poder especificar condiciones de búsqueda o que puedan parsear o traducir directamente los comandos del usuario;

en esta fase de diseño es mejor abstraerse del tipo de presentación que se hará. Una vez que la entidad de control tenga en su poder la condición de búsqueda, lanzará la convocatoria de acuerdo a los datos de la misma, es responsabilidad de las demás entidades saber si responden a esta convocatoria o no. La forma de respuesta de las entidades va a ser directamente generando reportes con la información pedida, pero si a cada entidad se le codifica la orden de generación de reportes, se tendrían tres codificaciones repetidas (una por entidad de datos). Para evitar esto se propone lo siguiente: que cada entidad responda si pertenece a la condición definida en la convocatoria; si la entidad cumple con la condición, la entidad de control incluirá a la entidad en turno en un conjunto que se denominara "conjunto de inclusión" (en esta fase de diseño, la estructura del conjunto y la representación de las entidades a incluir son irrelevantes y se pueden dejar como decisiones de implantación). Después, un módulo controlado por le entidad de control (o se podría definir alguna otra entidad, debido a que la entidad de control puede llegar a ser muy grande) se encargará de generar los reportes a partir del conjunto de inclusión.

De esta manera se evita que las entidades sean muy grandes y repetitivas. Esta forma de trabajo está ilustrada en la figura 4.3.

Siguiendo la filosofía orientada a procedimientos, la forma de funcionamiento sería la siguiente: el módulo de control se encarga de recibir la condición especificada por el usuario y de forma estructurada (de acuerdo a la fig 4.2) va relegando responsabilidades hasta llegar a un subprograma que se encarga directamente de accesar la tabla correspondiente y buscar la eneada que cumpla con la condición especificada. A partir de este punto se puede reconocer un problema paralelo al de generación de reportes en el sistema orientado a objetos: el subprograma que busque las eneadas que cumplan con cierta condición puede reportar directamente esas eneadas o puede mandar la información a un conjunto o algún otro tipo de estructura que servirá para guardar las eneadas que cumplan con la condición especificada. Si el subprograma generara directamente los reportes, se tendría que habría que incluir este código en todas las subrutinas que busquen eneadas y el sistema entero sería muy grande y lento. Si el subprograma sólo guarda la información de esas eneadas en un conjunto, al final solamente se necesitaría otro módulo que generara los reportes a partir de ese conjunto. Esta forma de trabajo está ilustrada en la figura 4.4.

Una vez diseñada la estrategia de funcionamiento, se puede concretar la forma física en que se realizará este funcionamiento, o el control que tendrá el usuario sobre el sistema para poderlo hacer funcionar de la manera que se ha definido (en el análisis y el diseño).

Para que los objetos puedan responder a una convocatoria dada, es necesario que el sistema reserve un compartimento, espacio, marco u opción para que la entidad de control pueda hacer la convocatoria, los objetos puedan responder e integrarse al conjunto de inclusión; es decir, un espacio para la selección de entidades de acuerdo a una condición o comando proporcionado por el usuario. De igual manera, el sistema reservará un área para que actúe la entidad de generación de reportes (si el sistema estará dividido en áreas de acción, será más

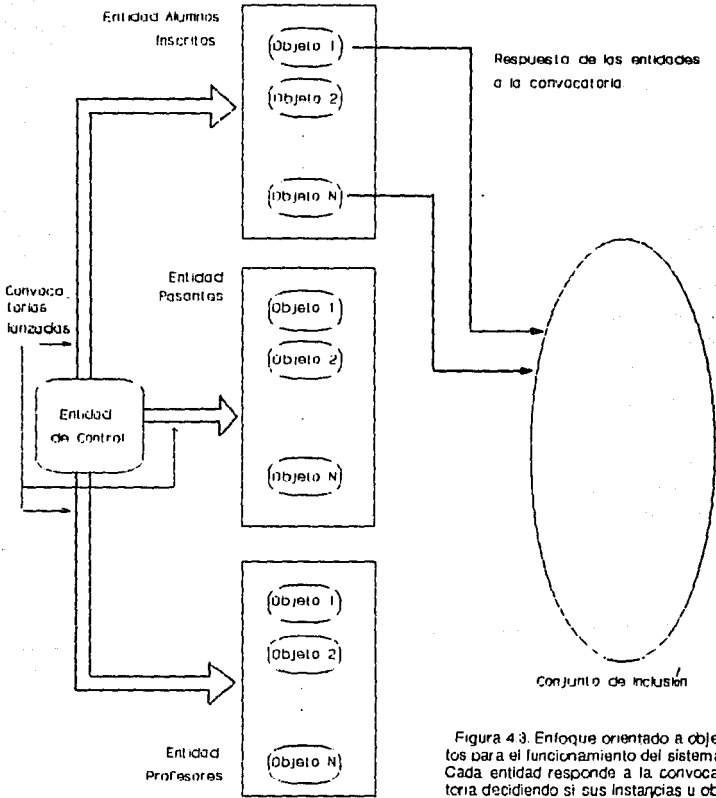


Figura 4.3. Enfoque orientado a objetos para el funcionamiento del sistema. Cada entidad responde a la convocatoria decidiendo si sus instancias u objetos cumplen con la condición definida en la convocatoria.

fácil definir una entidad de generación de reportes).
Entonces, se puede describir un mapa del sistema para que contenga, al menos, las siguientes opciones o comandos:

1. Transferencia de información (cargar, guardar)
2. Manipulación de información (agregar, editar, borrar)
3. Selección de entidades
4. Generación de reportes
5. Salir del sistema

La opción de transferencia de información se refiere a la posibilidad del sistema de poder manejar archivos persistentes. Cargar información se refiere a transferir información de un archivo permanente a una estructura de objetos en memoria, para que de esta forma se puedan manejar los datos. Guardar información se refiere al proceso contrario: a partir de la información en memoria, el sistema debe ser capaz de almacenarla en archivos persistentes. Cuando el sistema recibe la opción de salir, debe verificar si la información en memoria ha sido editada, en tal caso, deberá guardarla y después salir. La estructura de los archivos persistentes y la forma en que se va a transferir información, además de la estructura y organización de los objetos en memoria son detalles que se pueden resolver en una etapa posterior para la codificación, aunque a esta etapa de diseño pertenecen la aclaración y definición de las estrategias de transferencia de información y organización de objetos en memoria. Este último tema (organización de objetos) será cubierto en el siguiente apartado (4.2.3), la estrategia de transferencia de información se ilustra en la figura 4.5.

4.2.3. Lista de Objetos Candidatos.

Las entidades identificadas en la etapa del análisis del sistema definen tipos o clases de objetos (las clases son equivalentes, hasta cierto punto, de un tipo de dato o estructura de datos) que implican la existencia de instancias u objetos. Esto quiere decir que las entidades que se han definido solamente son las estructuras adyacentes de los objetos que se usarán. La fase de diseño de objetos significa que se deben enlistar los objetos que se usarán, se debe definir la estructura real de las clases a usar; la primera etapa de esta fase es la realización de los objetos que se usarán, la identificación de sus interrelaciones, jerarquías y descendencia.

Las entidades principales que se han identificado son:
Alumnos inscritos
Pasantes
Profesores

Las entidades Alumnos inscritos y la entidad Pasantes comparten la característica de que ambas entidades deben tener características de estudiantes. Por ejemplo, ambas entidades deben tener un campo para almacenar el número de cuenta. Al conjunto formado por las dos entidades (Alumnos inscritos y Pasantes) se les conocerá como Estudiantes. Los Estudiantes comparten con la entidad Profesores el hecho de que deben poseer datos acerca de su propia identidad (es decir, tanto los Estudiantes como los Profesores deben tener nombre, dirección, etc.), esto es, las entidades Estudiantes y Profesores -

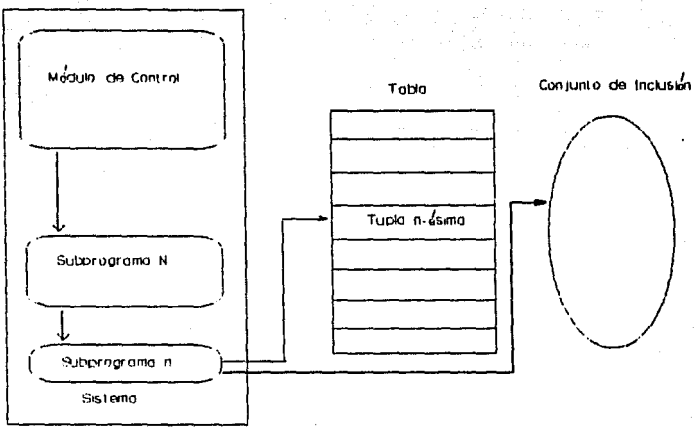


Figura 4.4 ¡Enfoque orientado a procedimientos para el funcionamiento del sistema. El sistema está integrado por el módulo de control y diversos subprogramas. Cuando el módulo de control recibe un comando del usuario, pasa el control al subprograma correspondiente (representado aquí como el subprograma N) para accesar cierta tabla o ciertos datos lógicos; a su vez, este subprograma llama a otro subprograma específico (representada como el subprograma n) para recuperar los datos pedidos, se accesa la tabla y se busca la tupla n-ésima que cumpla con las condiciones definidas por el usuario. El subprograma n guarda los datos de la tupla n-ésima encontrada en el conjunto de inclusión para que después se genere un reporte a partir de estos datos.

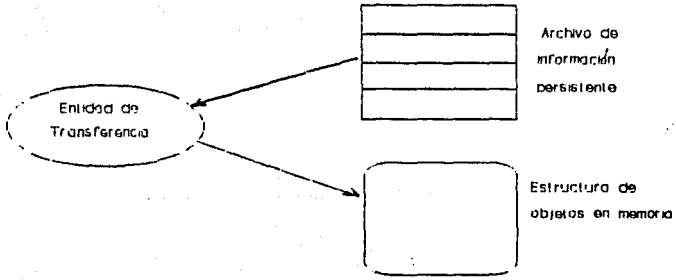
deben tener una estructura o conjunto común para los datos personales. Con estos datos, se puede realizar un diagrama de descendencia (a veces llamado diagrama familiar) de las entidades principales; este diagrama se muestra en la figura 4.6 y servirá de guía en la decisión de las características a heredar para cada entidad. Como estas entidades son las que contienen los datos imprescindibles y las rutinas de trabajo del sistema, es decir, estas entidades son las que hacen que el sistema funcione, se les llamará entidades Activas o entidades Actores, para diferenciarlas de las entidades que se definirán y que tienen una participación pasiva en el sistema, que sirven para arbitrar el flujo de información o de control en el sistema o que tienen algun tipo de interacción con el usuario.

En el apartado de diseño de transacciones se mencionó la transacción de edición de datos; esto implica la definición de una entidad de Edicion que tenga alguna forma de control sobre las variables de datos de las entidades Activas. Si esta entidad de Edición tuviera acceso a estas variables de las entidades Activas, se estaría violando el principio de encapsulación de información; para evitar este defecto de diseño, es necesario que las entidades Activas tengan rutinas para editarse a si mismas, la entidad de Edición solamente les mandaría los mensajes adecuados a las otras entidades, de acuerdo a los comandos del usuario. Esto es, la entidad de Edicion se limitara a presentar al usuario opciones de edición y a aceptar de éste comandos de edición; su relación con las otras entidades se limitará a controlar el flujo de control de edición de datos.

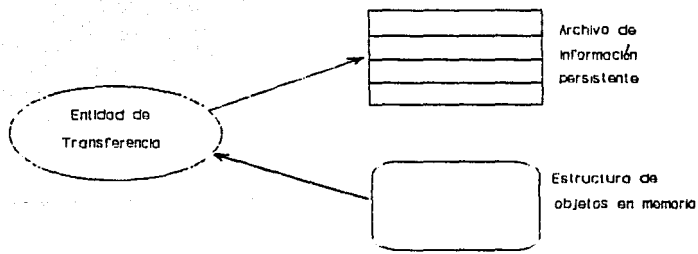
Las otras dos entidades identificadas son la entidad de Control y la entidad de Generación de Reportes. La entidad de Control deberá tener rutinas para presentación de las opciones para realizar las transacciones antes definidas, y para aceptar comandos del usuario. Se definió que una vez que la entidad de Control reciba un comando de usuario, debe lanzar una convocatoria. Esta convocatoria lanzada por la entidad de control no es un mensaje "al aire", es decir, no deberá ser un mensaje que se transmita a todas las entidades a ver quién contesta; el mensaje deberá estar dirigido a alguna entidad y esta se encargará de recolectar las respuestas de sus instancias (por ejemplo, si el usuario transmite el comando <<reportar los datos del profesor Pedro Gomez>>, la entidad de control no debe transmitir este mensaje a las entidades Estudiante porque sería inútil. La capacidad de la entidad de control para controlar los mensajes enviados a las otras entidades es a lo que se le ha llamado "convocatoria", porque cualquier entidad Activa podría responder en un momento dado).

La entidad de Generación de Reportes deberá tener métodos para accesar el conjunto de inclusión y desplegar la información contenida (aquí no se viola el principio de encapsulación puesto que la entidad de Generación de Reportes no accesa las variables de las instancias integrantes del conjunto). Como la entidad de Control va a desplegar la opción de Generación de Reportes y va a controlar los comandos del usuario, entonces la entidad de Control también controlará la participación de la entidad de Generación de Reportes.

Todas estas nuevas definiciones e interrelaciones modifican el diagrama de la figura 4.3; entonces, es necesario definir un diagrama más detallado. Este se muestra en la figura 4.7.



a) Carga



b) Guarda.

Figura 4.5. Estrategias de transferencia de información.
a) Carga. Transfiere información de un archivo persistente a una estructura de objetos.
b) Guarda. Transfiere información de una estructura de objetos a un archivo persistente.
Estas transferencias son hechas por una entidad de transferencia de información.

4.2.4. Refinamiento de Objetos.

Las primeras entidades que se deben refinar son las entidades Activas; en adelante estas entidades se denominarán clases Activas, estas clases servirán de referencia para definir instancias u objetos. En los diagramas de relaciones de entidades se mostraron diagramas de estas clases de forma que el concepto de entidad encerraba a sus instancias, es decir, cada clase engloba a sus objetos instanciados (es decir, definidos como una variable de esa clase). Algunos conceptos de base de datos ayudarán a clarificar los conceptos antes mencionados: en el análisis se identificó la necesidad de definir una entidad Profesores, si se piensa en la entidad Profesores como una estructura que puede contener los datos de un profesor, se tendrá como resultado que el sistema solamente podra guardar y administrar los datos de un profesor; en cambio, si se piensa en la entidad Profesores como una definición de tipo, se tendrá como resultado que el sistema podrá controlar todas las variables de tipo Profesores para que cada variable almacene los datos de un profesor.

Estos conceptos estan basados en la definición de estructuras de datos y tipos de datos, pero, en realidad, la definición de una clase Profesores (para seguir con el ejemplo) requiere definir también el comportamiento que tendrán las instancias de esa clase. Según el paradigma orientado a objetos, cada instancia u objeto debe contener datos y los métodos necesarios para manipular esos datos. Esto es, la clase profesores deberá definir las estructuras de datos necesarias para el almacenamiento de los datos de profesores, métodos (rutinas) para identificar, procesar y presentar esos datos de acuerdo a las sentencias enunciadas en el análisis de requerimientos, además, se deberán definir métodos para la edición de sus propios datos, según se estableció en el apartado anterior. Todo lo anterior es cierto para las otras clases Activas. Cada clase debe definir un esquema de definición o una cierta agrupación de tipos para que las instancias de esa clase ya sepan como controlarse o procesarse a si mismas. En forma global, estos conceptos se ilustran en la figura 4.8.

En esta figura se ilustra otro problema potencial: la forma en que se van a organizar las instancias. En la figura 4.8 se hace la propuesta que los objetos estén organizados en estructuras dinámicas, es decir, que cada objeto sea creado durante la ejecución del sistema y que apunte al siguiente objeto creado en el sistema.

Para resolver este problema se usarán algoritmos para el manejo de listas ligadas. Generalmente las listas dinámicas se administran mediante procedimientos: para integrar o agregar un elemento o nodo a la lista, para borrarlo, etc. La diferencia principal dentro del paradigma orientado a objetos radica en que cada objeto debe saber como integrarse y administrarse dentro de la lista; usando la herencia de propiedades entre objetos, estas caracteristicas pueden definirse en un objeto y después heredar las definiciones a los objetos Activos. Todas estas definiciones quedarán mejor explicadas mediante la figura 4.9.

Otro punto que se debe definir es la estructura que tendrá el conjunto de inclusion. Hasta el momento este conjunto está definido como una caja negra, como una estructura donde se "guardan" los objetos que responden afirmativamente a una convocatoria. Si la estructura del conjunto fuera algún tipo de lista dinámica donde sus nodos fueran los objetos mismos, se gastaria demasiado espacio de almacenamiento en información que estaria duplicada; por eso se propo-

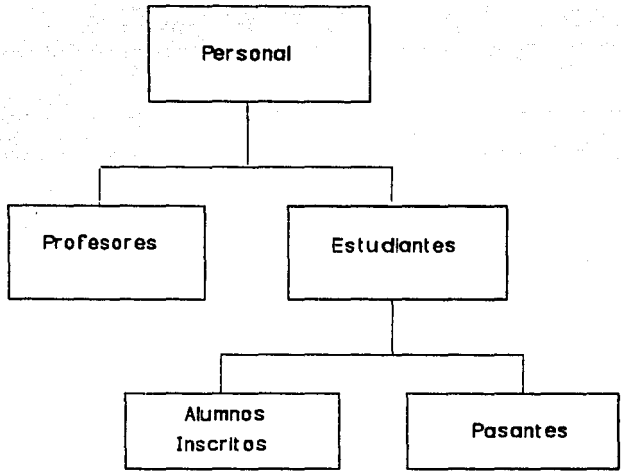


Figura 4.6. Diagrama de descendencia de las entidades identificadas en el análisis. En este diagrama se muestra la herencia de características comunes de los objetos candidatos.

ne el uso de una lista dinámica de apuntadores, tal como se muestra en la figura 4.10. De esta manera, solamente se señalan los objetos que respondieron afirmativamente a la convocatoria.

4.2.5. Diseño de Objetos.

Las clases Activas ya han quedado determinadas, así como las clases que tienen una participación de control. Estas últimas clases solamente necesitan tener definidos unos métodos de control de información, de paso y recepción de parámetros por lo que se definirán solamente las clases Activas (estas clases solamente se definirán para efectos de diseño).

La primera clase en el diagrama de descendencia fue el de administración de listas, en esta clase se deberán definir dos objetos, por lo menos, para tener un administrador de lista y un objeto que sirva de soporte para los nodos o elementos de esa lista. Esto es, la clase de administración de listas deberá definirse para tener:

Tipos de apuntadores;
Objeto lista

- Apuntador cabeza (de lista);
- Apuntador final (de lista);
- Apuntador auxiliar (para recorrer la lista) que se llamará cursor;
- Variable contador para llevar la cuenta del número de elementos en la lista;
- Métodos para: Borrar al primer nodo en la lista;
- Borrar al último nodo en la lista;
- Borrar al nodo apuntado por cursor;
- Leer información del nodo apuntado por cursor;
- Buscar un objeto en la lista;
- Buscar el siguiente objeto en la lista.

Objeto nodo

- Apuntador al siguiente nodo;
- Tamaño del nodo;
- Métodos para integrarse a una lista-parámetro.

La clase personal deberá estar compuesta como sigue:

- Variable nombre;

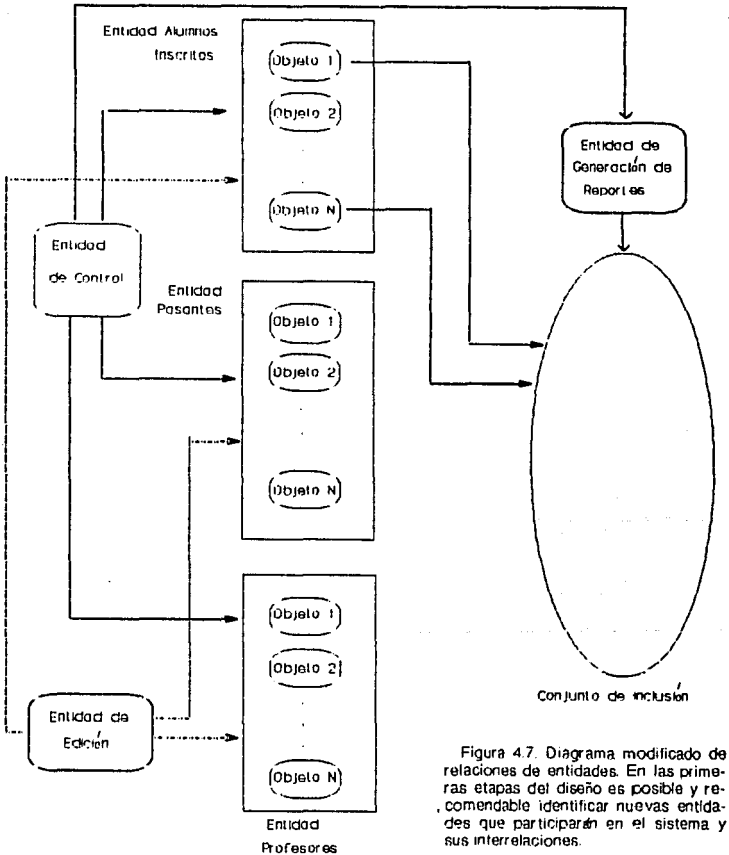


Figura 4.7. Diagrama modificado de relaciones de entidades. En las primeras etapas del diseño es posible y recomendable identificar nuevas entidades que participarán en el sistema y sus interrelaciones.

- Variable sexo;
- Variable teléfono;
- Método busca objeto a partir del nombre dado.

La clase estudiantes será:

- Variable no. de cuenta;
- Variable nombre de la carrera o licenciatura;
- Variable total de créditos;
- Variable promedio;
- Variable actividades extraescolares;
- Método para estimar si el alumno es considerado buen estudiante.

La clase alumnos inscritos se definirá:

- Variable créditos;
- Variable semestre actual;
- Variable número máximo de semestres;
- Variable número de materias en curso;
- Variable número de créditos en curso;
- Métodos para: Calcular y comparar el porcentaje de créditos;
- Estimar si un alumno está dentro de un periodo de tiempo legal según los estatutos universitarios;
- Estimar si el alumno es considerado regular;
- Recuperar el número de materias, el semestre y los créditos.

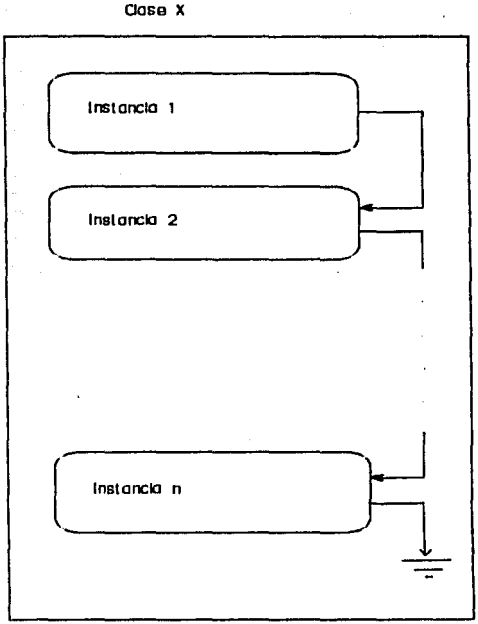


Figura 4.8. Esquema estructural de organización de objetos en memoria. La clase X se refiere a que cada entidad define una clase (de Alumnos Inscritos, de Pasantes o de Profesores), y cada clase puede tener un número n de instancias u objetos que se organizarán en una lista dinámica.

La clase pasantes es como sigue:

- Variable para almacenar el lugar donde se realizó (realizará) el servicio social;
- Variable para almacenar la fecha de inicio del servicio social;
- Variable para almacenar la fecha en que termina el servicio social;
- Variable para almacenar el proyecto o actividades en que se participó;
- Variable para almacenar el nombre del responsable o asesor del prestador del servicio social;
- Variable para almacenar el título de la tesis (si esta variable está en blanco, significa que no se ha registrado la tesis);
- Variable para almacenar la fecha de registro de la tesis;
- Variable para almacenar la fecha de vencimiento del registro;
- Variable para la fecha asignada para el exámen profesional (si esta variable está en blanco quiere decir que no se ha registrado el exámen profesional);
- Variable para guardar el nombre de los sinodales asignados;
- Métodos para: Verificar y recuperar todos los datos acreca del servicio social;
- Verificar y recuperar datos de la tesis;
- Verificar y recuperar datos del exámen profesional.

La clase profesores será definida como sigue:

- Variable para almacenar el sueldo;
- Variable para la profesión;
- Variable para guardar las áreas de conocimiento;
- Variable para el número de horas asignadas;
- Variable para el número de materias;
- Variable para el nombre de las materias;

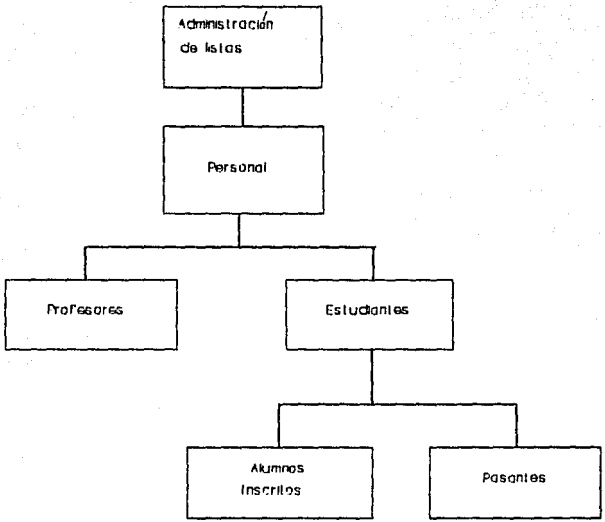


Figura 4.9 Nuevo diagrama de descendencia. Si se define un objeto para la administración de listas, y después se toma a este objeto como ancestro de los objetos Activos, se tendrá que todos los objetos necesarios heredarán las características de administración de listas definidas en el objeto ancestro.

- Comparar una cierta materia;
- Comparar una cierta profesión;
- Comparar un área de conocimiento;
- Recuperar el número de horas y materias.

El desarrollo y codificación de los algoritmos necesarios para realizar las tareas antes mencionadas (tanto para las clases Activas como para las clases de control de flujo de información) queda enmarcado dentro del campo de las técnicas de programación y por lo tanto no se discutirá dentro de este trabajo, aunque se presenta el diseño conceptual en el apéndice A, como referencia..

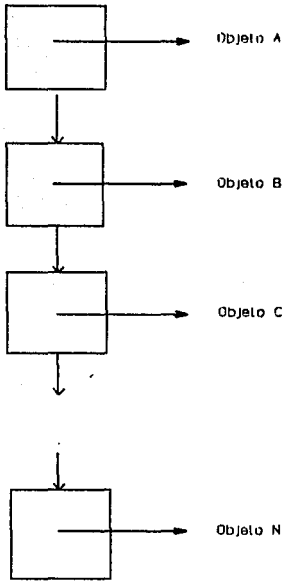


Figura 4.10. Estructura del conjunto de inclusión. Esta estructura consiste en una lista dinámica de apuntadores. Si a una cierta convocatoria responden i objetos, la lista se extenderá para abarcar i objetos, si contestaran j objetos, la lista se extendería para j objetos, por eso la lista debe ser dinámica. Cada nodo de esta lista será un apuntador al objeto que responda afirmativamente a la convocatoria; de esta manera, si el objeto A responde, la lista incluirá un apuntador al objeto A, de esta forma se puede accesar el conjunto de inclusión y ver cuales objetos están marcados para la impresión.

CAPITULO 5

Conclusiones y Trabajos Posteriores.

5.1. CONCLUSIONES.

El objetivo principal planteado al inicio de este trabajo fue mostrar el diseño o forma de funcionamiento de un sistema de administración de bases de datos (DBMS), este objetivo se cumplió totalmente puesto que se explicaron los fundamentos de las DBMS, además de las diferentes filosofías o formas de funcionamiento para una aplicación de este tipo.

El análisis desarrollado para dos sistemas de bases de datos quedó enmarcado dentro de los objetivos propuestos. Este análisis también se cumplió puesto que se planteó un problema aplicable dentro de los límites del campo de las bases de datos; este problema se analizó y se planteó su resolución bajo las perspectivas de los paradigmas orientado a procedimientos y orientado a objetos.

El desarrollo orientado a objetos está sustentado por el marco de referencia mostrado a lo largo del capítulo 2. Las técnicas descritas en este capítulo fueron aplicadas en el desarrollo mostrado en la segunda parte del capítulo 4 (4.2).

5.2. TRABAJOS POSTERIORES.

Los objetivos perseguidos por esta tesis se enfocaron al análisis y el diseño de un sistema usando técnicas orientadas a objetos. Un posible seguimiento a los conceptos presentados a lo largo de este trabajo, sería continuar con las fases de desarrollo del ejemplo propuesto en esta tesis; es decir, enfocarse en los detalles de implantación y programación de la solución propuesta.

Se puede desarrollar un trabajo de seguimiento a mediano plazo enfocandose más a las fases posteriores del ciclo de vida de un sistema, se puede pensar en su optimización/modificación y en su mantenimiento, destacando si las técnicas orientadas a objetos usadas en el diseño son realmente de utilidad en el sentido de mantenimiento del sistema.

Aún después de la realización de las fases del ciclo de vida del sistema, se puede plantear la integración del sistema a un medio ambiente computacional enfocado a los datos que se pueden usar o al desarrollo de este medio ambiente a partir del sistema actual.

APENDICE A

Pseudocódigo de los métodos principales usados por el sistema.
En este apéndice se presentan los algoritmos generales para la codificación de los métodos de las clases Activas.

------- Objeto lista -------

Método para borrar el primer nodo en la lista;
si el contador de elementos en la lista es mayor a cero,
(existen elementos en la lista)
entonces, quitar el nodo apuntado por el apuntador cabeza;
si el contador de lista es igual a 1,
(existe un solo elemento en la lista)
los apuntadores cabeza, final y cursor deben apuntar a NIL;
de lo contrario,
(hay más de un elemento en la lista)
si el apuntador cursor es igual al apuntador cabeza,
(cursor apunta al primer elemento, a borrar)
entonces, cursor debe apuntar al siguiente elemento en la lista;
de lo contrario,
(cursor apunta a algun otro nodo, por lo tanto, no tiene caso moverlo)
cabeza debe apuntar al siguiente elemento en la lista;
(fin de decision, si cursor = cabeza)
(fin de decision si contador = 1)
decrementar el contador de lista;
de lo contrario,
(hay cero elemento en la lista, no se puede borrar)
desplegar mensaje y anular la operación de borrado;
(fin de decisión, si contador > 0)

Método para borrar el ultimo nodo en la lista;
si el contador de elementos en la lista es mayor a cero
(existen elementos en la lista)
entonces, si el contador de lista es igual a 1,
(existe un elemento en la lista)
quitar el nodo apuntado por final;
los apuntadores cabeza, final y cursor deben apuntar a NIL;
de lo contrario,
(hay más de un elemento en la lista)
poner al apuntador cursor igual a cabeza;
mientras el campo del nodo apuntado por apuntador cursor para apuntar al siguiente nodo sea diferente al apuntador final;
mover el apuntador cursor al siguiente elemento;
(fin de ciclo de recorrido)
(en este punto, cursor apunta al penultimo elemento de la lista)
quitar el nodo apuntado por final;
poner el campo del nodo apuntado por cursor para apuntar al siguiente nodo a NIL;
poner final igual a cursor;
(fin de decisión si contador = 1)
decrementar el contador de lista;
de lo contrario,
(hay cero elemento en la lista, no se puede borrar)
desplegar mensaje y anular la operación de borrado;

```
(fin de decisión, si contador > 0)

   Método para borrar el nodo apuntado por cursor;
si el contador de elementos en la lista es mayor a cero
 (existen elementos en la lista)
entonces, si el contador de lista es igual a 1,
  (existe un elemento en la lista)
  quitar el nodo apuntado por cursor;
  los apuntadores cabeza, final y cursor deben apuntar a NIL;
 de lo contrario,
  (hay más de un elemento en la lista)
  si cursor es diferente a cabeza,
   (cursor apunta a un nodo que no es el primero)
   poner un apuntador local igual a cabeza;
   mientras el campo del nodo apuntado por el apuntador local para
apuntar al siguiente nodo sea diferente al apuntador cursor;
    mover el apuntador local al siguiente elemento;
   (fin de ciclo de recorrido)
   (en este punto, el apuntador local apunta al elemento anterior al
nodo apuntado por cursor)
   si cursor es igual a final,
    entonces, poner final igual al apuntador local;
    poner el campo del nodo apuntado por final para apuntar al
siguiente nodo a NIL;
   de lo contrario,
    poner el campo del nodo apuntado por el apuntador local para
apuntar al siguiente nodo igual al campo del nodo apuntado por cursor
para apuntar al siguiente nodo
   (fin de decisión cursor = final)
  de lo contrario,
   (cursor es igual a cabeza)
   mover cabeza al siguiente nodo;
  (fin de decisión cursor = cabeza)
  quitar el nodo apuntado por cursor;
 (fin de decisión si contador = 1)
 decrementar el contador de lista;
de lo contrario,
 (hay cero elemento en la lista, no se puede borrar)
 desplegar mensaje y anular la operación de borrado;
(fin de decisión, si contador > 0)

   Método para leer la información del nodo apuntado por cursor;
si el contador de elementos en la lista es mayor a cero;
 entonces, desplegar o guardar en un registro la información del nodo
apuntado por cursor
 de lo contrario,
  en su caso, anular el registro que contendría la información del
nodo;
(fin de decisión, si contador > 0)

   Método para buscar un objeto en la lista;
poner cursor igual a cabeza;
llamar al método para buscar el siguiente objeto en la lista;
(fin de rutina)
```

Método para buscar el siguiente objeto en la lista;
inicializar dos variables locales para indicar si ya se encontró el nodo (encontrado) y si estamos al final de la lista (alfinal). Estas variables se inicializaran al valor booleano falso;
utilizar una variable de tipo procedimiento y asignarle la dirección de un método que se debe pasar como parámetro;
este método-parámetro contendrá las instrucciones acerca de cual información especifica se va a buscar, y de que tipo de objeto)
si el contador de objetos en la lista es mayor a cero y cursor es diferente a NIL,
entonces,
(el nodo apuntado si existe)
mientras las variable locales encontrado y alfinal sean igual a falso;
si la variable procedimiento con parámetro cursor regresa un valor verdadero, entonces,
poner la variable encontrado a verdadero;
de lo contrario,
si el campo del nodo apuntado por cursor para apuntar al siguiente nodo es diferente a NIL,
entonces avanzar cursor para apuntar al siguiente nodo;
de lo contrario, (cursor apunta al ultimo elemento)
poner la variable alfinal a verdadero;
(fin de decisión, si cursor^.siguiente <> NIL)
(fin de decisión, si daemon(cursor) = true)
(fin de ciclo, mientras (encontrado = false) and (alfinal = false))
regresar valor de encontrado o desplegar mensaje, según sea el caso;
de lo contrario, (la lista está vacia o ya se buscó)
regresar un valor nulo o desplegar mensaje, según sea el caso;
(fin de decisión, si (contador > 0) and (cursor <> NIL))

Nota acerca de la nomenclatura de diseño: el término daemon se refiere a cualquier variable tipo procedimiento que puede referirse a cualquier método segun se asigne.

----- Fin de objeto lista -----

------- Objeto nodo -------

Método para integrarse a una lista-parámetro;
si el apuntador cabeza de la lista es igual a NIL, entonces
(la lista está vacia)
los apuntadores cabeza y final de la lista deben ponerse al mismo valor que el apuntador al nodo que se va a integrar a la lista;
de lo contrario,
(hay otros nodos en la lista)
poner el campo del nodo apuntador por final para apuntar al siguiente nodo igual al valor del apuntador al nodo a integrarse a la lista;
poner el apuntador final igual al valor del apuntador del nodo a integrarse;
(fin de decisión, si cabeza = NIL)
poner el campo del nodo para apuntar al nodo siguiente a NIL;
incrementar el contador de elementos en la lista;

----- Fin de objeto nodo -----

Nota: todos los métodos de las clases siguientes se refieren a los

campos del nodo apuntado por el apuntador cursor;

------- Clase personal -------

Método para buscar información almacenada en un nodo;
comparar la cadena parámetro con la cadena en el campo correspondiente (puede ser según una clave-parámetro) del nodo;
regresar verdadero si las cadenas son iguales o falso si no lo son;
(nota: es necesario regresar algún valor booleano para compatibilidad con la variable tipo procedimiento definida en el método para buscar un nodo)

----- Fin de clase personal -----

------- Clase estudiantes -------

Método para estimar si el alumno es considerado buen estudiante;
poner el valor de una variable <<buen estudiante>> al valor verdadero si: el valor en el campo promedio es mayor o igual a 8.5 y si la cadena en el campo actividades extraescolares es diferente a cadena vacía;
regresar ese valor;

----- Fin de clase estudiantes -----

------- Clase alumnos inscritos -------

Método para calcular el porcentaje de créditos;
poner la variable porcentaje de créditos igual a número de créditos en turno sobre total de créditos; (campo heredado)
multiplicar el porcentaje de créditos por 100; (opcional)
regresar el valor de la variable;

Método para estimar si un alumno está dentro del tiempo legal;
poner la variable tiempo legal igual al valor lógico semestre actual igual al valor redondeado de número maximo de semestres multiplicado por 0.5;
regresar el valor de tiempo legal;

Método para estimar si un alumno es regular;
poner la variable regular igual al valor lógico promedio es mayor o igual a 6 y si el valor regresado por el método tiempo legal es verdadero;
regresar el valor de regular;

Método para recuperar el número de materias, el semestre actual y los créditos en curso;
llamar al método para leer la información del nodo apuntado por cursor para los campos:
número de materias en curso, semestre actual y número de créditos en curso;
regresar el valor de las variables usadas;

----- Fin de clase alumnos inscritos -----

------- Clase pasantes -------

Método para verificar y recuperar los datos del servicio social;
llamar al método para leer la información del nodo apuntado por cursor
para el campo del nombre del lugar donde se realizo el servicio
social;
si el valor regresado por el método es diferente a cadena vacía,
entonces,
llamar al método para leer la información del nodo apuntado por
cursor para los campos:
fecha de inicio de servicio social, fecha de terminación de servicio
social, nombre del proyecto o actividades en que se participó, nombre
del responsable o asesor;
regresar el valor de las variables utilizadas;
de lo contrario,
(no está registrado el servicio social)
desplegar mensaje de que no está registrado el servicio social;
regresar un valor nulo;
(fin de decisión, si lugar de servicio social <> " ")

Método para verificar y recuperar datos de la tesis;
llamar al método para leer la información del nodo apuntado por cursor
para el campo del titulo de la tesis;
si el valor regresado por el método es diferente a cadena vacía,
entonces,
llamar al método para leer la información del nodo apuntado por
cursor para los campos:
fecha de registro de la tesis, fecha de vencimiento del registro;
regresar el valor de las variables utilizadas;
de lo contrario,
(no está registrada la tesis)
desplegar mensaje de que no está registrada la tesis;
regresar un valor nulo;
(fin de decisión, si titulo de la tesis <> " ")

Método para verificar y recuperar datos del exámen profesional;
llamar al método para leer la información del nodo apuntado por cursor
para el campo de la fecha asignada para el exámen profesional;
si el valor regresado por el método es diferente a cadena vacía,
entonces,
llamar al método para leer la información del nodo apuntado por
cursor para el campo:
nombre de los sinodales asignados;
regresar el valor de las variables utilizadas;
de lo contrario,
(no está registrado el exámen profesional)
desplegar mensaje de que no está registrado el exámen profesional;
regresar un valor nulo;
(fin de decisión, si fecha de exámen profesional <> " ")

----- Fin de clase pasantes -----

------- Clase profesores -------

Método para recuperar el sueldo;

llamar al método para leer la información del nodo apuntado por cursor
para el campo del sueldo;
desplegar o regresar este valor;

Método para recuperar y comparar una materia;
llamar al método para leer la información del nodo apuntado por cursor
para el campo de nombre de las materias;
comparar la cadena pasada como parámetro con la cadena devuelta por
este último método;
si son iguales, entonces
desplegar un mensaje de "materia encontrada" o devolver un valor
verdadero;
de lo contrario, desplegar un mensaje diciendo que no se encontró la
materia o devolver un valor de falso;

Método para comparar una cierta profesión;
llamar al método para leer la información del nodo apuntado por cursor
para el campo de profesión;
comparar la cadena pasada como parámetro con la cadena devuelta por
este último método;
si son iguales, entonces
desplegar un mensaje de "profesión encontrada" o devolver un valor
verdadero;
de lo contrario, desplegar un mensaje diciendo que no se encontró la
profesión o devolver un valor de falso;

Método para comparar una cierta área de conocimiento;
llamar al método para leer la información del nodo apuntado por cursor
para el campo de área de conocimiento;
comparar la cadena pasada como parámetro con la cadena devuelta por
este último método;
si son iguales, entonces
desplegar un mensaje de "área de conocimiento encontrada" o devolver
un valor verdadero;
de lo contrario, desplegar un mensaje diciendo que no se encontró el
área de conocimiento o devolver un valor de falso;

Método para recuperar el número de horas y materias;
llamar al método para leer la información del nodo apuntado por cursor
para los campos;
número de horas asignadas y número de materias;
desplegar o devolver estos valores;

APENDICE B

Parámetros físicos del sistema.

Dado que no existe una codificación real y tangible del sistema, se podría pensar que no es posible establecer los parámetros físicos de uso del sistema. Una de las ventajas de la fase de diseño del sistema es que se puede hacer una extrapolación idealizada de las condiciones y necesidades de uso de un sistema; de esta manera se pueden jerarquizar todas las etapas de desarrollo de un programa, aún en el aspecto físico. Siguiendo estos lineamientos, la fijación real de los parámetros de un sistema se relegan a la fase de codificación o programación del sistema; el establecimiento conceptual de parámetros, o el establecer parámetros en una fase puramente de diseño tiene la ventaja de poderse abstraer de las técnicas de programación usadas en un momento dado o de las decisiones de programación que se tengan que hacer (uno de los objetivos de la ingeniería de software).

Con esta aclaración, se da paso al simple listado de los parámetros extrapolados para el sistema actual de base de datos:

En el apartado 4.2.5 (Diseño de Objetos) quedó establecida la forma en que se mapearían al dominio del software las entidades abstractas. A partir de estas consideraciones, se puede tomar un promedio de cuatro variables y cuatro métodos por cada objeto, es decir, aproximadamente:

20 bytes/variable = 80 bytes/variables de objeto
1K/método ≈ 4Kbytes/métodos de objeto
por lo que se concluye:
tamaño por objeto ≈ 4k

pero este tamaño es solamente para objetos de un solo nivel en un árbol de jerarquías (o diagrama familiar), es decir, sin considerar la herencia. Como los objetos Actores (Profesores, Pasantes y Alumnos Inscritos) heredan características, su tamaño aumentará. El objeto profesores hereda características de un solo antecesor (Persona), por lo que se deduce:

tamaño objeto profesores ≈ 8k

pero los objetos Pasantes y Alumnos Inscritos heredan características de dos ancestros (o generaciones) que les anteceden, por lo que:

tamaño objetos Pasantes/Alumnos Inscritos ≈ 12k

Si suponemos que en la fase de codificación se sigue un esquema de paginación, con un supuesto de 10 objetos en memoria a la vez:

lista de objetos profesores ≈ 80K
listas de objetos Pasantes/Alumnos Inscritos ≈ 120K c/una
tamaño necesario para los datos ≈ 320K

El código para un sistema como el propuesto no es muy extenso (dependiendo de las técnicas de codificación y el lenguaje empleado), por lo que cabe suponer:

tamaño del programa principal ≈ 20K

Considerando el espacio necesario para otros objetos (de Manejo de 1 Conjunto de Inclusión, de Edición, etc.), se puede suponer:

tamaño objetos no-Activos ≈ 30K

Es decir, en una primera aproximación o suposición en cuanto al tamaño completo del sistema se puede suponer:

tamaño mínimo = 370K

Otro punto importante que se debe considerar es la velocidad que necesita el sistema para que el sistema funcione dentro de un límite no prohibitivo en cuanto a las demoras de la computadora al usuario. Es necesario que este tema sea muy aproximado porque aún en la fase de programación es inconcebiblemente difícil hacer cálculos exactos en cuanto a este tópico. Sería necesario saber exactamente cuantas instrucciones simples forman el sistema, y saber cuanto tarda el procesador en realizar cada instrucción simple, además de ser necesario el dato del tiempo de acceso a memoria y la velocidad de interacción de la computadora.

Unicamente es necesario saber que la computadora tendrá que tratar con el manejo de objetos y con sus interacciones, a veces complejas, además de tener que procesar las interfaces interiores en el programa. Con estas consideraciones en mente, se puede concluir que una máquina de 8 MHz podría ser suficiente para el manejo del sistema.

BIBLIOGRAFIA

ATW90 - Atwood, Thomas; "Applying the Object Paradigm to Databases"; Computer Language; vol. 7, no. 9; Septiembre 1990.

BAI89 - Bailey, Stephen C.; "Designing with Objects"; Computer Language; vol. 6, no. 1; Enero 1989.

BOO87 - Booch, Grady; Software Tools with Ada: Structures, Tools & Subsystems; The Benjamin/Cummings Publishing Company; 1a. Edición, 1987.

BUL89 - Bulman, David M.; "An Object-Based Development Model"; Computer Language; vol. 6, no. 8; Agosto 1989.

BUL91 - Bulman, David M.; "Refining Candidate Objects"; Computer Language; vol. 8, no. 1; Enero 1991.

CON90 - Constantine, Larry L.; "Objects, Functions and Program Extensibility"; Computer Language; vol.7, no. 1, Enero 1990.

CHA89 - Chan, Alberto; Cátedras impartidas para la materia Computadoras II; ENEP Aragón, UNAM; 1989 - 1990.

COX90 - Cox, Brad J.; "There IS a Silver Bullet"; BYTE; vol. 15, no. 10; Octubre 1990.

EZZ90 - Ezzell, Ben; "Writing Reusable Objects"; Computer Language; vol. 7, no. 6; Junio 1990.

GIL88 - Gillenson, Mark L.; Introducción a las Bases de Datos; McGraw-Hill; 1a. Edición, 1988.

KHO90 - Khoshafian, Setrag & Abnous, Razmik; Object Orientation: Concepts, Languages, Databases, User Interfaces; John Wiley & Sons Inc.; 1a. Edición, 1990.

LAN90 - Lane, Alex; Object-Oriented Turbo Pascal; M&T Books, 1a. Edición, 1990.

PRE87 - Pressman, Robert; Software Engineering: A Practitoner's Approach; McGraw-Hill International Editions, 2a. Edición, 1987.

SUY90 - Suydam, Bill; "Smalltalk as a Programming Environment"; Computer Language; vol. 7, no. 4; Abril 1990.

SWA89 - Swan, Tom; Mastering Turbo Pascal 5.5; Hayden Books; 3a. Edición, 1989.