UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

DESARROLLO DE UNA APLICACION USANDO LA METODOLOGIA OMT

T E S I S

Que para obtener el Título de:

A C T U A R I O

p r e s e n t a:

ALEJANDRA OLMOS SILICEO

Dir. de Tesis:

M. en C. Guadalupe Ibargüengoitia G.

México, D. F. Septiembre de 1995

VNIVERSIDAD NACIONAL
AVPNºMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

"Desarrollo de una aplicación usando la metodología CMT"

realizado por Alejandra Olmos Siliceo

con número de cuenta 9157090-9 **, pasante de la carrera de** Actuaría

Dicho trabajo cuenta con nuestro voto **aprobatorio.**

Atentamente

Director de Tesis Propietario	M. C. Ma. Guadalupe E. Ibargüengoitia González
Propietario	Dra. Hanna Oktaba
Propietario	M. C. Gustavo Manuel Flores
Suplente	Mat. Ana Luisa Solís
Suplente	Fis. Juan Jesús Gutiérrez

Consejo Departamental de Matemáticas

A Dios por haberme permitido culminar esta etapa de mi vida satisfactoriamente.

A mis padres por haber forjado los cimientos y el carácter para llegar a este punto del camino.

A Mary por ser mi héroe de niñez y estar siempre a mi lado. Y al cariño de Rafael.

A Javi por hacerme sentir que soy alguien importante en su vida.

A Lupita no sólo por asesorarme sino por tener el don de motivar a la gente y en tan poco tiempo por sus valiosos consejos.

A mis maestros por haber sembrado y cultivado en mí, la semilla del conocimiento.

A mis verdaderos amigos por siempre haber estado conmigo en momentos difíciles

¡GRACIAS!

Tabla de Contenidos

Introducción

El objetivo de este trabajo es mostrar a través de un caso práctico una metodología Orientada a Objetos (OO) para el desarrollo de sistemas llamada "Técnica de Modelado de Objetos" u OMT por sus siglas en inglés (Object Modeling Technique), la cual, fué creada por un grupo de investigadores del centro de desarrollo e investigación de General Electric, en Estados Unidos.

Se optó por este objeto de estudio por lo innovador que éste representa, además por las numerosas aplicaciones que actualmente existen bajo esta etiqueta.

Haciendo resaltar que el paradigma OO es considerado como una nueva manera de enfocar los problemas, usando básicamente conceptos del mundo real y no los computacionales. El presente trabajo consta de tres capítulos, donde se muestra la utilización de los conceptos Orientados a Objetos durante todo el ciclo de vida de la ingeniería de software, desde el análisis hasta la implementación de manera clara y concisa dando un caso práctico como ejemplo.

En el primer capítulo se da un panorama general del paradigma Orientado a Objetos empezando con sus antecedentes históricos. Posteriormente, se tratan las características de esta filosofía de ingeniería de software, así como la definición de los conceptos generales que la rigen, pues estos se estarán utilizando constantemente en los capítulos posteriores.

El capítulo dos muestra el marco teórico de la metodología OMT así como su notación y los pasos a seguir para el desarrollo de software a través de las etapas de análisis y diseño. También muestra el diseño de bases de datos relacionales como implementación del análisis y diseño OO realizado.

Posteriormente el capítulo tres muestra un caso práctico de esta técnica de desarrollo de software que se inicia desde el planteamiento del problema, seguidas por las etapas

de análisis, diseño e implementación, dando como resultado una aplicación funcionando, la cual, está almacenada en el diskette adjunto.

Finalmente se enlistan las conclusiones a las que se han llegado una vez que ha sido realizado el trabajo.

I. Paradigma Orientado a Objetos

En este capítulo se dan los antecedentes históricos que propiciaron el surgimiento del paradigma Orientado a Objetos. También se describen las características por las que se rige esta filosofía, así como la diferencia principal que existe con el paradigma tradicional (estructural).

Posteriormente se definen sus conceptos generales que se utilizarán a lo largo del presente trabajo, esto con el fin de ubicar al lector en este paradigma junto con dichos conceptos.

I.1. *Historia*

En la historia de la Computación, la cual se considera relativamente joven, el desarrollo de software ha sido menos acelerado que el de hardware. Así pues, en el transcurso del tiempo la estructura de los programas se ha hecho más compleja y por tal razón complica el proceso de modificar o actualizar dichos programas. Todos estos problemas aunados a la poca reutilización de piezas de software en nuevos sistemas ha provocado una búsqueda constante de una mejor estructuración y expresividad en el desarrollo de software.

Uno de los resultados de dicha búsqueda es el paradigma Orientado a Objetos (OO). Otra de las causas que motivaron el origen de la OO es la evolución acelerada de los lenguajes de programación. En los años 60's, se dió a conocer el lenguaje ALGOL-60, el cual dió la pauta al desarrollo de la metodología de la Programación Estructurada. Este lenguaje marcó el inicio de una nueva generación de lenguajes de programación estructurados.

En ésta misma época, en Noruega, un grupo de investigadores, encabezados por Dahl y Nygaard, buscaban un cambio en los lenguajes existentes para hacerlos más cercanos a los problemas del mundo real mediante aplicaciones "cotidianas". Se dieron cuenta que los problemas que debían simular estaban llenos de objetos que interactuaban entre sí, y aprovechando la buena

estructura de ALGOL-60, incrementaron los conceptos de objetos, clases de objetos y jerarquías de herencias entre clases. Precisamente éstos conceptos caracterizan hoy en día la Programación Orientada a Objetos. Así fué como nació SIMULA-67, el precursor de los lenguajes OO, aunque desafortunadamente no tuvo un impulso suficiente para seguir adelante, debido a que nació en un país pequeño, alejado de grandes centros de investigación y de los apoyos de la industria de las computadoras. [Oktaba-93a]

Fué hasta finales de los años 70's que se retomaron las ideas nacidas en SIMULA, pero ahora por un grupo fuerte de los Laboratorios Xerox, de Palo Alto, CA, en Estados Unidos, bajo la dirección de Alan Kay, en el proyecto conocido como Dynabook, cuyo objetivo era crear una interfaz muy amigable para una computadora personal y fué como se desarrolló el diseño e implementación del lenguaje SMALLTALK. Este lenguaje retomó los conceptos de objeto, clase y herencia de SIMULA, y los conjugó con la flexibilidad de la programación interactiva y libre de tipificación del lenguaje LISP. Con la popularización de este lenguaje, se comenzó a hablar del término Orientado a Objetos, dando lugar a una difusión muy acelerada del nuevo paradigma.

La segunda mitad de los 80's y los 90's se distinguen por una gran cantidad de propuestas de nuevos lenguajes orientados a objetos, de los cuales algunos se quedan simplemente como propuestas en el papel, pero otros se llevan rápidamente a la comercialización. Entre los más importantes se pueden mencionar:

- La extensión de C, conocida como C++, que permite el manejo de clases, de objetos y facilita todas las características pedidas en un lenguaje Orientado a Objetos.

- El más apegado de la herencia de SIMULA, es EIFFEL de Meyer, diseñado con base en los principios avanzados del manejo de objetos con énfasis en desarrollar software confiable a partir de componentes verificables y reusables.

- Una extensión Orientada a Objetos de LISP, conocida como CLOS (Common Lisp Object System). Resultó del trabajo de un grupo cuyo objetivo era la estandarización de LISP. Este grupo estudió las varias extensiones de LISP Orientadas a Objetos, ya existentes, pero en vez de seleccionar una de ellas como estándar, el grupo decidió desarrollar un nuevo lenguaje.

[Oktaba-93a] Oktaba Hanna, "¿Por qué está de Moda la Orientación a Objetos" Sistemas de Agosto 93.

- Cabe mencionar las distintas modificaciones de PASCAL, como OBJECT PASCAL o recientes versiones de TURBO PASCAL, que cuenta con un ambiente OO, pero no ha sido formulada un definición formal de sus características.

- Hay muchos intentos de combinar el paradigma Orientado a Objetos con la programación concurrente, paralela y distribuida, cuyo ejemplo puede ser el lenguaje ACTORS.

Actualmente la Orientación a Objetos ha adquirido gran popularidad originada por los lenguajes de programación, que ha hecho de su ámbito de aplicación muy amplio, pues lo podemos encontrar en distintas áreas de la Computación como son las Bases de Datos Orientadas a Objetos, en la programación, en redes, sistemas abiertos, sistemas heterogéneos basados en objetos y distintas herramientas para la ingeniería de software bajo la etiqueta OO.

I.2. *Filosofía General*

Históricamente, el enfoque central del desarrollo de software era plantear acciones o procedimientos ordenados para dar solución a problemas. A ésta perspectiva es la que se conoce como descomposición funcional o programación estructurada.

Este paradigma se considera ineficiente en sistemas complejos debido a que normalmente se trabaja con una abstracción en la que se debe tomar en cuenta la implementación, esto es, los sistemas se ven desde el punto de vista de la máquina y no de la aplicación misma. Esto provoca una estructura de software compleja y aún más cuando se trata de darle mantenimiento.

En contraste, la filosofía general del Análisis Orientado a Objetos se basa en abstracciones cercanas al mundo real sin preocuparse en la fase de análisis por su implementación. Esto es, una nueva manera de pensar para resolver problemas, ya que se ven como una colección de objetos que interactúan entre sí, haciendo el análisis de los problemas más cercanos al mundo real.

Esta nueva perspectiva se lleva a cabo identificando sustantivos u objetos de una aplicación. Los objetos se consideran entidades activas, capaces de efectuar operaciones, es decir, los objetos incorporan tanto su estructura a través de variables, como su comportamiento

mediante operaciones (figura 1.1). Una puerta de madera, el carro de Jorge, una lista de clientes son ejemplos de objetos.

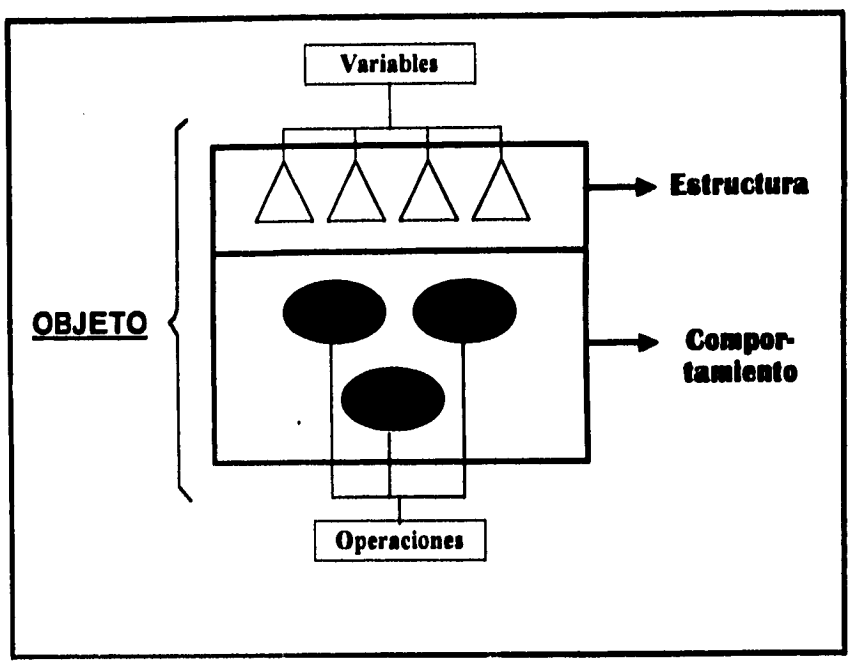


Figura 1.1

Es por ello que el desarrollo de software bajo éste paradigma resulta ventajoso. Además, de que los datos se encapsulan con sus operaciones en objetos, permite tener una jerarquización de ellos. Es decir, permite tener objetos generales y otros particulares donde los últimos heredan la estructura y acciones de objetos que se encuentran a un mayor nivel, permitiendo así un tipo de reutilización de código, el cual representaba un problema en la programación estructurada. Otro tipo de reutilización se efectúa cuando se utilizan objetos ya implementados para una aplicación en otra, promoviendo así la creación de bibliotecas que al paso del tiempo se enriquecerán. Aunque, para tener bibliotecas confiables es necesario crear objetos lo suficientemente generales y eficientes.

Una característica más del paradigma es que soporta clases de objetos, esto es, aquellos objetos que comparten su estructura y acciones pertenecen a una clase. Cabe mencionar que a pesar de su similitud se diferencían uno del otro. Esto se vé más claro en el objeto menú de acciones y el menú de opciones, ya que ambos pertenecen a la clase menú y uno es distinto al

otro. Por tanto, en un sistema en el que intervengan varios objetos de una clase, se estará reutilizando código, debido a que sus estructuras y funciones son iguales.

Otra ventaja que ofrece el paradigma es que evita que los usuarios tengan acceso a la estructura interna de los objetos. De esta manera no se corre el riesgo de que se modifique sin un suficiente conocimiento para ello, aumentando así la confiabilidad de los sistemas.

En la ingeniería de software, una vez que se ha desarrollado un sistema, éste sufrirá modificaciones o actualizaciones con el fin de darle mantenimiento. La capacidad de considerar a los objetos como entidades con un grupo de acciones facilita el mantenimiento de los sistemas.

En términos generales la Orientación a Objetos es un paradigma para el desarrollo de software, el cual consiste en una colección de objetos en comunicación que incluyen tanto su estructura como su comportamiento.

I.3. *Conceptos Generales*

En ésta sección se definirán aquellos conceptos que caracterizan al paradigma Orientado a Objetos con el fin de evitar confusiones y para un mejor entendimiento de la terminología que se utilizará a lo largo del trabajo.

- Objeto

  Un objeto es una entidad que se caracteriza por tener estado, comportamiento e identidad. El estado de un objeto está dado por un conjunto de datos variables que describe su estructura interna. Su comportamiento se refiere a una serie de procedimientos o métodos que puede llevar a cabo un objeto. Y por último, la identidad se refiere a aquello que lo distingue de todos los demás, en otras palabras es aquello que lo hace único.

  No únicamente se considera como objeto a aquellos que son concretos ó tangibles del mundo real, como por ejemplo persona, bicicleta, etc., sino también a los conceptuales, es decir, los que no existen realmente pero que intervienen en la

aplicación para darle solución, como un cedulador que se encargue de enviar ciertos mensajes cada período.

El objeto es precisamente la característica base del DOO ya que alrededor de ellos y su comunicación entre sí, constituyen el punto de vista para resolver los problemas.

- **Clase de Objetos**

Una clase de objetos se define como la abstracción de un conjunto de objetos que comparten su estado y comportamiento. Por ejemplo, puede haber listas de clientes, listas de alumnos, listas de maestros, etc., y todos estos objetos pertenecen a la clase lista ya que poseen estructura y comportamiento semejante.

Normalmente se refieren a clases de objetos con la palabra clase. Un objeto que pertenece a una clase recibe también el nombre de instancia de esa clase.

El estado de una clase va a depender directamente tanto de la aplicación como del criterio del desarrollador. Por ejemplo, los estados que interesan de la clase motor en una aplicación pueden ser encendido y apagado, y en otra caliente o frío.

- Atributos

Los atributos son las características que definen el estado de un objeto en cualquier momento. Por ejemplo, nombre, edad y peso son atributos de la clase persona.

Es importante marcar la diferencia entre identificadores internos y atributos del mundo real. Los identificadores internos no proporcionan ningún tipo de significado en el dominio del problema y solo se trata de un convenio para la implementación. En cambio, los atributos del mundo real si son significativos para la aplicación.

- **Método**

Un método es una operación o función que pueden realizar las instancias de una clase.

Por ejemplo alguno de los métodos de la clase lista son: insertar, borrar elementos, dar el primer elemento, etc.

- **Polimorfismo**

Polimorfismo es la característica que permite que una operación se comporte de distinta manera en clases distintas. Cada objeto sabe exactamente como llevar a cabo sus propias operaciones, por lo mismo, el usuario de una operación no necesita saber como se realiza sino únicamente se interesa en el resultado de ésta.

Un ejemplo claro de polimorfismo son las figuras geométricas, ya que el usuario no se preocupa de la manera en que se dibuja un círculo, un rectángulo y una línea, solamente le interesa que se dibujen.

- Herencia

La herencia es una de las características más importante del paradigma OO, ya que permite la relación jerárquica entre clases, es decir, soporta tener clases generales y a partir de ellas, definir clases más particulares que heredan los métodos y variables (estado) de la clase general. La clase general recibe el nombre de superclase de aquellas clases que se encuentran a un nivel menor. Las clases particulares que constituyen el refinamiento de una(s) clase(s) general(es) reciben el nombre de subclases.

Cabe mencionar que una instancia de una subclase es también una instancia de su superclase.

Existen dos tipos de herencia:

- **Herencia Sencilla**
- **Herencia Múltiple**

Decimos que se trata de herencia sencilla cuando una clase hereda propiedades solamente de una superclase.

Existe herencia múltiple cuando una clase tiene más de una superclase. Esto permite mezclar información de dos ó más fuentes, aumentando así la reutilización de software.

- **Asociación**

Asociación es la relación entre dos ó más clases. Según el número de clases que se relacionen reciben el nombre de asociación binaria (entre dos clases), ternaria (entre tres clases), etc. Aunque se recomienda que se intente descomponer las asociaciones entre 3 ó más clases en asociaciones binarias con el fin de disminuir la complejidad del análisis.

Como caso especial de asociación es la agregación, la cual consiste en relacionar el todo con sus partes. En otras palabras, cuando un conjunto de objetos representa las partes ó componentes de algo, y asociándose éstos con un objeto que representa el todo.

La agregación posee dos propiedades:

- *Propiedad Transitiva.-* **Esto significa que si A es parte de B y B es parte** de C entonces A es parte de C.

- *Propiedad Antisimétrica.-* **Si A es parte de B entonces B no es parte de** A.

II. Metodología de Desarrollo de Software

El presente capítulo muestra al lector básicamente la metodología OMT, su notación y el proceso de desarrollo de software que se utilizará para el desarrollo de la aplicación. Además, muestra el marco teórico del diseño de bases de datos relacionales como implementación del análisis y diseño orientado a objetos a realizar.

II.1. *Proceso de Desarrollo de Software*

Durante los primeros años de la era de las computadoras, el desarrollo de software se realizaba virtualmente sin ninguna planificación, ya que la mayoría de éste se elaboraba y utilizaba por la misma persona u organización. La misma persona lo escribía, lo ejecutaba y si fallaba lo depuraba por lo que normalmente no existía la documentación.

Posteriormente se introdujeron conceptos de programación estructurada, que originaron mayores niveles de sofisticación tanto de hardware como de software. Llegaron las "casas de software", las cuales se caracterizaban por "desarrollar el último paquete de software" y comercializarlos.

De ésta manera crecieron el número de sistemas y bibliotecas de software, los cuales, sufrían modificaciones (por fallas o adaptaciones según las necesidades) que se convirtieron en una actividad muy costosa en cuanto a tiempo y esfuerzo.

A medida que se incrementaban tanto el número de sistemas informáticos como su complejidad, se desarrollaron varias técnicas para el desarrollo de software, con el fin de estructurarlo u organizarlo, dando origen a la programación estructurada y posteriormente al paradigma orientado a objetos.

La idea principal de la programación estructurada, es un paradigma funcional, es decir, toma en cuenta que operaciones se necesitan realizar y en que orden se ejecutan para dar solución a un problema. Una de sus principales desventajas es que al querer hacer un cambio o modificación a un sistema desarrollado bajo este paradigma puede resultar complicado y muy costoso. A pesar de esto, se han desarrollado varias metodologías estructuradas como los métodos orientados al flujo de datos, los orientados a la estructura de datos, desarrollo de sistemas de Jackson, etc.

En cambio, el paradigma orientado a objetos crea sistemas alrededor de los objetos del mundo real o los conceptuales, lo que facilita modificaciones futuras y la reutilización de código. Entre las metodologías bajo éste paradigma se encuentra el método de Booch, la metodología OMT (Técnica de Modelado de Objetos), así como otros.

Independientemente del paradigma que se utilice, el proceso de desarrollo de software consta de un ciclo de vida clásico, el cual, consiste en el análisis, diseño, codificación, prueba y mantenimiento.

En el análisis del sistema se busca establecer los requerimientos de todos los elementos del sistema y asignar parte de éstos al software, ya que en ocasiones el software se interrelaciona con otros elementos como personas, bases de datos y/o hardware.

En el análisis de los requerimientos del software, como su nombre lo indica su enfoque central es el software. En esta etapa el analista debe comprender el dominio de la información así como las funciones, rendimiento e interfaces requeridas.

El proceso de diseño traslada los requerimientos del software a un conjunto de representaciones que describen la estructura de datos, arquitectura y procedimiento algorítmico. El proceso de diseño, al igual que los requerimientos, se documenta con el fin de que cualquier persona pueda entenderle.

La codificación es la etapa que sigue del diseño, la cual consiste en traducir las representaciones de la etapa anterior a un lenguaje de máquina. La realización de un diseño detallado puede hacer de la codificación más sencilla y mecánica.

La etapa de prueba consiste en asegurar que el sistema realiza lo que se espera de él, además que las entradas definidas produzcan los resultados esperados.

El software tendrá cambios después que se entregue al cliente, ya sea, para corregir errores, o para realizar adaptaciones o modificaciones para reforzar el sistema. Por ello, se aplica el mantenimiento a cada una de las etapas anteriores del sistema ya existente en vez de empezar uno nuevo.

Las etapas que anteriormente se describieron de una manera general, constituyen el ciclo de vida clásico para la ingeniería de software. Es importante mencionar que el método en cada una de las etapas puede variar de un paradigma a otro, pero el enfoque global de cada etapa permanece invariable. Además, cabe resaltar que cada una de ellas debe estar documentada con el fin de evitar confusiones.

II.2. *Metodología Orientada a Objetos para el Desarrollo de Software (OMT)*

[Oktaba-93c], resume los criterios para elegir una buena metodología Orientada a Objetos, basándose en la comparación de 23 métodos de análisis y diseño. Los criterios se describen en los siguientes puntos:

1. Debe proporcionar herramientas para la definición de las vistas estáticas de un sistema. Para ello se necesitan reglas de identificación y colocación de clases semánticas, sus atributos y sus relaciones de generalización, agregación y otras. Para expresar las vistas estáticas se requiere de una notación gráfica que sea entendible y fácil de utilizar. También es deseable tener contemplado, en el diseño, la inclusión de las clases de aplicación, utilerías e interfaz.

2. Debe proporcionar herramientas y notación para la definición del comportamiento dinámico del sistema -vistas dinámicas. Esta parte es especialmente importante cuando el sistema que estamos diseñando es distribuido, paralelo y/o tiene requerimientos de sistemas de tiempo real.

3. Debe ayudar a manejar la complejidad del sistema a nivel estructural, y no estaría mal si lo hiciera también a nivel dinámico.

[Oktaba-93c] Oktaba Hanna, "Análisis y Diseño Orientado a Objetos" Soluciones Avanzadas de Julio-Agosto 93.

En base a éstos puntos se decidió utilizar la metodología "Técnica de Modelado de Objetos" (OMT), que se enfoca en los objetos que intervienen en la aplicación y no en las funciones del sistema. Al igual que otras metodologías, cuenta con las etapas de análisis, diseño e implementación, aunque las fronteras entre una fase y otra no son tajantes, pues se va desarrollando el sistema de manera iterativa. Cabe mencionar que no existen discontinuidades debido a que la notación en las distintas etapas es la misma. La utilización de éste método requiere un mayor esfuerzo en el análisis, pero el resultado del diseño es más claro y adaptable, permitiendo realizar cambios futuros con mayor facilidad.

En la etapa de análisis, su objetivo es entender y modelar la aplicación y el dominio en el que opera, enfocando los tres principales aspectos del sistema: los objetos y sus relaciones; el flujo dinámico de control y los cambios funcionales de datos.

El resultado de ésta etapa es la creación de tres modelos que no son completamente independientes, pero tienen la característica de que cada uno puede ser examinado y entendido por sí mismo. Dichos modelos son:

- Modelo de Objetos
- Modelo Dinámico
- Modelo Funcional

II.2.1. Notación

Como toda metodología de desarrollo de software, cuenta con una notación. Cabe resaltar que ésta notación se mantiene presente durante todo el desarrollo debido a que éste se lleva a cabo de manera iterativa aunque la notación únicamente ilustra los tres modelos resultantes de la etapa de análisis.

A continuación se define la notación de estos trea modelos dando una explicación de los distintos aspectos que cada uno trata definir independientemente de la aplicación que se trate.

- ***Modelo de Objetos***

El modelo de objetos describe la estructura estática de los objetos en un sistema, es decir, sus identidades, sus relaciones con otros objetos, sus atributos y operaciones.

Se representa gráficamente con diagramas de objetos que definen las clases que intervienen en el sistema junto con su estructura y comportamiento, así como las asociaciones con otras clases.

Las clases se representan encerrando el nombre de ésta, sus valores y sus métodos en un rectángulo separando con una línea estos tres aspectos, como lo muestra la figura 2.1.

Una asociación es una relación entre dos o más clases por lo que puede ser binaria, ternaria o de mayor grado, dependiendo del número de clases que asocie. Aunque la mayoría de las asociaciones se pueden reducir a binarias. Las figuras 2.2 y 2.3 muestran las representaciones de las asociaciones binarias y ternarias respectivamente.

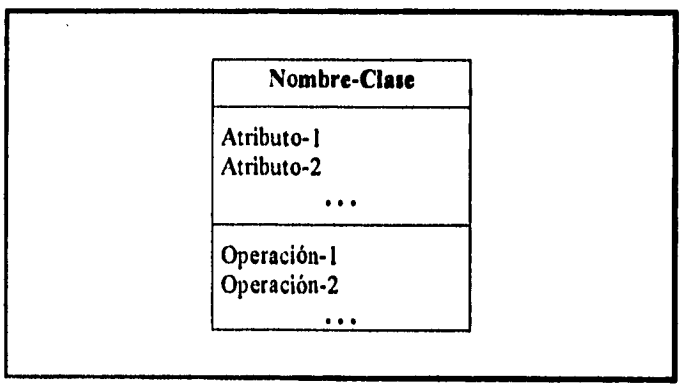


Figura 2.1

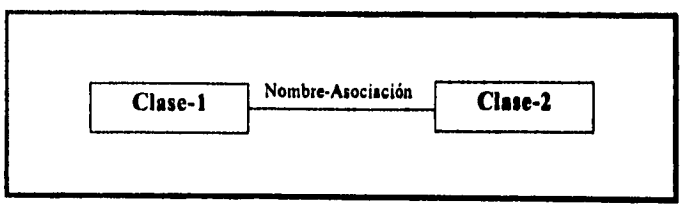


Figura 2.2

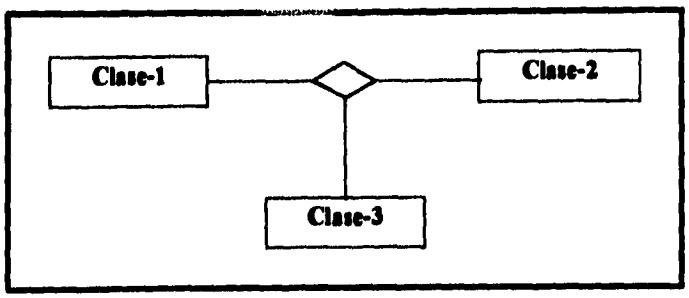


Figura 2.3

Un caso especial de asociación es la agregación, que relaciona un todo y sus partes donde el todo está formado de las partes. Por ejemplo, una computadora que representa el todo está formada por CPU, teclado, mouse y unidad de disco, donde éstos representan sus partes. La notación de la agregación se ilustra en la figura 2.4.

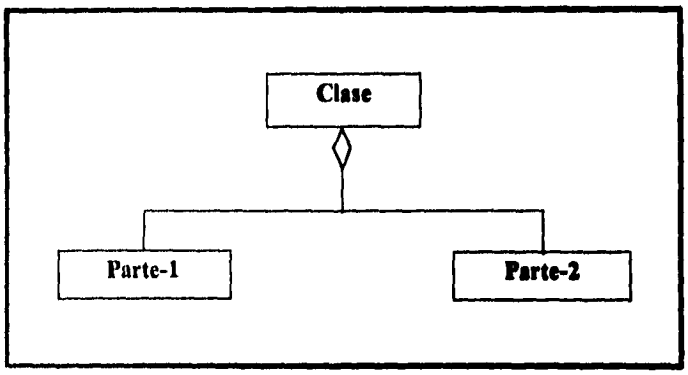


Figura 2.4

En el modelo de objetos se especifica también la multiplicidad de las asociaciones, es decir, cuantas instancias de una clase se relacionan con cada instancia de otra clase. Las representaciones de las distintas multiplicidades de asociaciones se muestran en la figura 2.5.

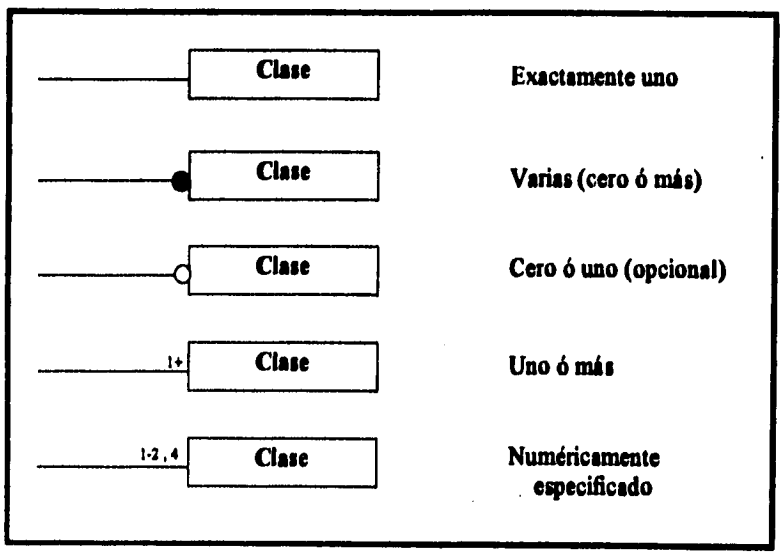


Figura 2.5

Por último, la representación de la herencia de clases se muestra en la figura 2.6.

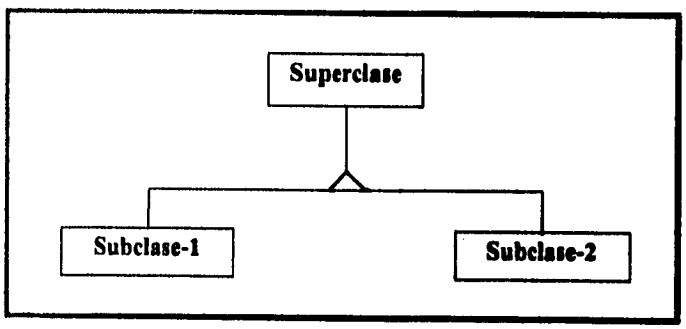


Figura 2.6

- ***Modelo Dinámico***

El modelo dinámico describe aquellos aspectos de un sistema que tienen que ver con el tiempo y con la secuencia de operaciones. Esto se refiere a eventos que marcan cambios, las secuencias de eventos, los estados que definen el contexto de eventos y la organización de eventos y estados. Los eventos son los estímulos externos y los estados se refieren a los distintos valores que pueden tomar los objetos.

El modelo dinámico captura el control de un sistema, sin tomar en cuenta que hacen las operaciones, sobre que operan ni como están implementadas.

El modelo dinámico está representado gráficamente mediante un conjunto de diagramas de estados. Cada diagrama muestra los estados y secuencias de eventos permitidos en un sistema, por una clase de objetos. Los estados se representan con círculos y los eventos con flechas (figura 2.7).

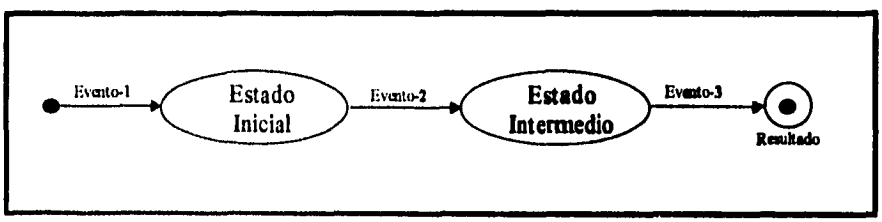


Figura 2.7

- *Modelo Funcional*

El modelo funcional describe las transformaciones de valores, esto es, como se derivan los datos de salida en el cálculo de un proceso a partir de los datos de entrada sin tomar en cuenta el orden en que los valores son calculados, ni como, ni porque se realizan.

Este modelo se representa gráficamente por diagramas de flujos de datos, que muestran las dependencias entre valores y el cálculo de datos de salida. Los procesos se representan mediante círculos. Las flechas que llegan a un proceso se consideran datos de entrada y los datos de salida se representan con flechas que salen de un proceso (figura 2.8).

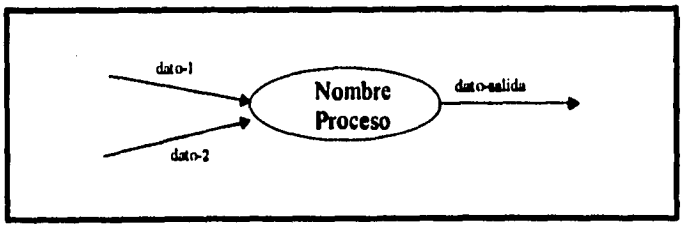


Figura 2.8

En estos diagramas intervienen también objetos actores y pasivos. Los objetos actores son aquellos que producen y consumen información, los cuales se representan con rectángulos (figura 2.9); y los objetos pasivos únicamente guardan información para un acceso posterior representados por líneas paralelas horizontales (figura 2.10).

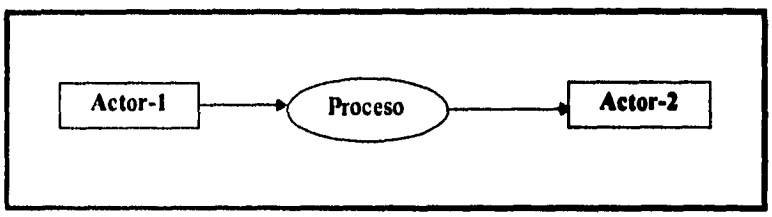


Figura 2.9

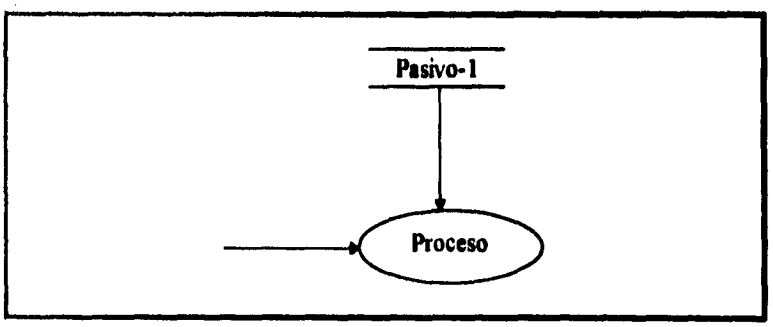


Figura 2.10

Durante el diseño de un sistema se determina la arquitectura de éste. Con ayuda del modelo de objetos se organiza el sistema en subsistemas, las decisiones se derivan a partir del modelo dinámico y las prioridades haciendo negociaciones, es decir, balanceando las decisiones para hacerlo eficiente.

II.2.2. Análisis

El primer paso del análisis es hacer una descripción del problema dando un panorama general del sistema. La descripción no necesariamente debe ser completa y formal, además no se debe considerar como inmutable, pues avanzar en el análisis lo hace más claro y disipa ambigüedades e inconsistencias. Para ello, el analista debe estar en comunicación con el cliente. La descripción debe contener los requerimientos y debe especificar que cosas son obligatorias y cuales opcionales.

- ***Modelo de Objetos***

Como siguiente paso es la construcción del modelo de objetos, para su realización la información se obtiene de la descripción del problema, de conocimientos del dominio de la aplicación y de conocimientos del mundo real. Para su desarrollo es conveniente seguir los siguientes pasos:

- *Identificación de Clases*

De la descripción del problema, se extrae una lista de candidatos a clases de objetos a partir de los sustantivos y se añaden otras que se identifiquen en el dominio de la aplicación o del mundo real. Posteriormente se descartan:

⇒ Clases redundantes (aquellas clases que expresan la misma información)

⇒ Clases irrelevantes (tienen que ver poco o nada con el problema)

⇒ Clases vagas (las clases deben ser específicas)

⇒ Atributos (nombres que se puedan considerar como atributos de una clase)

⇒ Operaciones (nombres que describen métodos)

⇒ **Roles (el nombre de una clase debe reflejar su naturaleza y no el papel que juega en una asociación)**

⇒ **Construcción de implementación (pueden necesitarse después, en este nivel de análisis no)**

- *Preparar un diccionario de datos*

Para evitar confusiones es necesario escribir un pequeño párrafo que describa a cada clase de objetos.

- *Identificar asociaciones entre objetos*

Cualquier dependencia entre dos o más objetos es una asociación. Para hacer más sencilla la identificación de asociaciones entre objetos, se escriben frases verbales de la descripción del problema. A partir de ésta selección, se descartan:

⇒ Aquellas que asocien **clases ya eliminadas**

⇒ Asociaciones irrelevantes o de **implementación**

⇒ Acciones (una asociación **debe describir una propiedad estructural del** dominio de la aplicación)

⇒ Asociaciones ternarias **(la mayoría de ellas se pueden descomponer en** binarias)

⇒ Asociaciones derivadas **(ocasionan redundancia puesto que pueden ser** definidas en términos de **otras asociaciones)**

Posteriormente, se debe **dar nombre a las asociaciones, añadir roles donde se** considere necesario para saber el **papel que juega una clase con respecto a otra, y especificar la** multiplicidad.

- *Identificar atributos*

Los atributos son propiedades de los objetos, es decir, los valores que pertenecen a un objeto. Para la identificación de atributos, generalmente se hace uso del conocimiento de la aplicación y del mundo real.

- *Refinar usando herencia*

Se utiliza la herencia para organizar las clases y para hacer más simple la estructura de las clases, considerando que puede ser en dos direcciones:

⇒ Generalizando aspectos comunes de varias clases a una superclase

⇒ Refinando clases existentes en subclases

La primera dirección en la aplicación de herencia, es un poco más natural de realizar, ya que los atributos y métodos comunes de varias clases van a formar parte de la superclase.

La herencia hacia abajo, esto es, mediante refinamiento no es tan inmediata. Frecuentemente, la manera de ubicar donde se puede aplicar es a través de planteamientos de varios casos del problema a resolver. Por ejemplo, si en alguna aplicación se hablan de figuras geométricas y lo que se busca en ese problema involucra a círculos y cuadrados, entonces se aplicaría el refinamiento.

- *Verificar que existan caminos para posibles preguntas*

En éste punto se cuenta ya con un modelo de objetos preliminar, por tal razón, una manera de saber si falta considerar algunos aspectos, es mediante el planteamiento de preguntas y ver si pueden responderse siguiendo una trayectoria en dicho modelo.

- *Iterar y refinar el modelo*

Al continuar con el análisis, en caso de encontrar con una deficiencia es necesario regresar a una etapa anterior para corregirla en caso de ser necesario. Por tal razón se dice que el análisis es un proceso iterativo.

- ***Modelo Dinámico***

Una vez construido el modelo de objetos se sigue con el desarrollo del modelo dinámico. El análisis dinámico se va a enfocar en el comportamiento dinámico de los objetos que intervienen en el sistema. Este análisis se inicia mediante la observación de eventos, estímulos visibles y respuestas. Para la construcción de éste modelo se siguen los siguientes pasos:

- *Preparar escenarios*

Un escenario es una secuencia de eventos, los cuales, ocurren siempre que haya cambio de información entre objetos. Para conocer mejor el comportamiento de un sistema, es conveniente preparar diálogos típicos entre usuario(s) y el sistema, incluyendo tanto casos normales como casos de error.

- *Identificar eventos entre objetos*

Mediante la examinación de los escenarios construidos, se pueden identificar los eventos (cambio de información de un objeto a otro), ubicando a los objetos emisores y receptores. Los objetos intercambiando eventos se pueden mostrar en una traza. Una traza es un diagrama que muestra los objetos en líneas verticales y los eventos en líneas horizontales que unen al objeto emisor con el receptor; y de manera secuencial se ubican los eventos haciendo esto más simple con ayuda de los escenarios. El tiempo en la traza avanza de arriba hacia abajo.

- *Construir el diagrama de estados*

Únicamente se debe construir diagramas de estados para aquellas clases que tienen un comportamiento dinámico significativo, mostrando el patrón de eventos que recibe y manda, junto con las acciones que la transforma.

El diagrama de estados de una clase se construye con ayuda de las trazas que involucren dicha clase, ya que los estados corresponden a las flechas que salen de la traza y las acciones son las flechas que llegan. Posteriormente, se verifica que exista consistencia entre los distintos diagramas de estados.

El conjunto de diagramas de estados constituyen el modelo dinámico del sistema.

- ***Modelo Funcional***

El modelo funcional muestra la dependencia entre valores y las funciones que los relacionan. Los procesos en el diagrama de flujo de datos corresponden a actividades o acciones en los diagramas de estados, por esto, es conveniente construir el modelo funcional después de haber desarrollado los otros dos modelos. Para su construcción se siguen los siguientes pasos:

- *Identificar valores de entrada y de salida*

Los valores de entrada y de salida son parámetros de eventos entre el sistema y el mundo real. Se examina la descripción del problema para encontrar cualquier valor de entrada o de salida que haya faltado.

- *Construir los diagramas de flujo de datos mostrando dependencias funcionales*

El diagrama de flujo de datos muestra como a partir de valores de entrada se calculan los valores de salida. Se considera más sencillo identificar las funciones que generan cada valor de salida, que identificar todos los usos de cada valor de entrada.

- *Describir las funciones*

Cuando el diagrama de flujo de datos se haya refinado lo suficiente, es conveniente escribir una descripción de cada función, la cual, puede ser en lenguaje natural, pseudocódigo, ecuaciones matemáticas, etc.

- *Identificar restricciones*

Las restricciones se refieren a dependencias funcionales entre objetos que no se relacionan mediante valores de entrada o de salida. Las precondiciones de una función son restricciones que los valores de entrada deben satisfacer y las postcondiciones son restricciones que deben cumplir los valores de salida.

II.2.3. Diseño

El enfoque en la etapa de análisis es conocer o definir que se necesita hacer sin tomar en cuenta como se va a hacer. La etapa de diseño busca como se va a resolver el problema, inicialmente de manera superficial y a medida que se avanza en él, se van incrementando los niveles de detalle.

La etapa de diseño está constituida por el diseño del sistema y el diseño de objetos.

El diseño del sistema se desarrolla antes que el de objetos, ya que en éste, el diseñador toma decisiones como la estructura de alto nivel del sistema, la partición del sistema en subsistemas designándolos a hardware o software, constituyendo de esta manera la arquitectura del sistema.

En el diseño de objetos se agregan detalles de implementación tales como reestructuración de clases para aumentar eficiencia, definir estructuras de datos internas y algoritmos para implementar operaciones, todo esto con el fin de hacer casi mecánica la implementación del sistema.

- *Diseño del Sistema*

En el diseño del sistema se define la arquitectura de éste. El modelo de objetos marca la pauta para organizar el sistema en subsistemas, después se organiza la concurrencia ubicando grupos de objetos en tareas concurrentes. Se toman decisiones de comunicación, de almacenamiento de datos y de la implementación del modelo dinámico, como

qué subsistemas se implementarán en hardware, cuáles en software, la comunicación entre subsistemas, etc. Finalmente las prioridades se establecen mediante negociaciones.

Para esto, es necesario que el diseñador tome las decisiones que se enlistan a continuación.

- *Organizar el sistema en subsistemas*

El primer paso del diseño del sistema es dividir éste en varias partes, donde cada una recibe el nombre de subsistema. Cada subsistema agrupa aspectos del sistema con alguna propiedad en común, como la misma ubicación física, funcionalidad semejante o ejecución en el mismo tipo de hardware.

Un subsistema normalmente se puede identificar a través del servicio que aporta. Un servicio es un conjunto de funciones para llevar a cabo un propósito.

Cada subsistema tiene una interfaz bien definida al resto del sistema, permitiendo así, que cada uno se pueda diseñar de manera independiente sin que afecte a los otros.

- *Identificar concurrencia inherente en el problema*

Tanto en el análisis como en el mundo real todos los objetos son concurrentes. Sin embargo en una implementación no sucede esto, ya que un procesador puede soportar algunos objetos. Por esta razón, se debe identificar que objetos deben estar activos concurrentemente y cuales mutuamente exclusivos, pues los objetos que no necesitan estar activos al mismo tiempo, bien pueden estar trabajando en un sólo procesador.

Es de gran utilidad el modelo dinámico para poder identificar concurrencia, ya que dos objetos son inherentemente concurrentes si ambos pueden recibir eventos al mismo tiempo sin interactuar.

- *Designar subsistemas a procesadores y tareas*

Para que el diseñador pueda tomar esta decisión, es necesario que estime los cálculos necesarios y los recursos para satisfacerlos.

El usar varias unidades de hardware es con el fin de obtener mayor desempeño que una sola puede dar. El número de procesadores va a depender del número de cálculos y de la velocidad de la máquina.

Posteriormente, el diseñador debe decidir que subsistemas deben implementarse en hardware y cuales en software. Para ello se debe considerar la compatibilidad, costo y problemas de desempeño. También se debe tomar en cuenta la flexibilidad para futuros cambios.

Después se designan tareas de varios subsistemas de software a procesadores, ya que algunas tareas necesitan un lugar físico específico o porque la proporción de cálculos es muy grande para un único procesador.

Los subsistemas que más interactúan entre sí, deben estar en un mismo procesador para reducir los costos de comunicación. Y los subsistemas independientes se designan a procesadores separados.

Finalmente se determina la conectividad de las unidades físicas que implementan a los subsistemas.

- *Escoger la estrategia básica para la implementación del almacenamiento de información*

El almacenamiento de información puede implementarse a través de estructuras de datos, archivos y/o bases de datos.

Los archivos pueden ser sencillos, baratos y permanentes pero necesita mucha programación adicional para manejarla. En cambio, las bases de datos dan un mayor nivel de abstracción que los archivos pero involucran compromisos en cuanto a costos y complejidad.

- *Identificar recursos globales y determinar mecanismos para controlar los accesos a ellos*

El diseñador debe identificar los recursos globales tales como unidades físicas, espacio, nombres lógicos y acceso a la información. También es necesario determinar mecanismos para controlar los accesos a los recursos globales. Alguno de los mecanismos más comunes son establecer un objeto "guardián" que controla todos los accesos, o bien, dividir los recursos globales en subconjuntos que se manejen a un nivel menor.

- *Seleccionar una aplicación para implementar el control de software*

El diseñador debe escoger entre varias maneras de implementar el control de software. Aunque no existe una necesidad lógica para que todos los subsistemas usen la misma implementación, es común que decida usar un único estilo de control para todo el sistema.

El control de software puede ser en el manejo de procedimientos, en el manejo de eventos y en la concurrencia. El control para el manejo de procedimientos en un sistema reside en el programa donde se decide escribir el código. En cuanto al control en un sistema para el manejo de eventos reside en el monitor. Finalmente para controlar la concurrencia en un sistema es a través de varios objetos independientes.

- *Considerar condiciones de frontera*

A pesar de que el mayor esfuerzo del diseño en varios sistemas radica en un comportamiento ideal, el diseñador debe también considerar condiciones de frontera como inicialización, terminación y fallas.

- *Establecer prioridades de negociación*

Un aspecto esencial de la arquitectura de un sistema es hacer consideraciones entre tiempo y espacio, hardware y software, simplicidad y generalidad, y eficiencia y mantenibilidad. Estas consideraciones van a depender de la aplicación misma.

- ***Diseño de Objetos***

Los objetos que se identificaron en el análisis sirven como esqueleto para el diseño, aunque el diseñador debe escoger entre varias maneras de implementarlos tratando de minimizar tiempo de ejecución, memoria y costos.

En el diseño de objetos, todas las operaciones que se identificaron en el análisis se expresan como algoritmos y las clases, atributos y asociaciones se deben implementar como estructuras de datos bien definidas.

Toda la información que aparece en el análisis de alguna manera debe estar presente en el diseño. La manera más sencilla es hacer del diseño un proceso de refinamiento de los modelos de la etapa de análisis.

El modelo funcional describe las operaciones que debe implementar el sistema. En el diseño se decide como se debe implementar cada operación, escogiendo un algoritmo y descomponiendo operaciones complejas en operaciones más sencillas siempre pensando en que sea fácil implementarse, que sea entendible y optimizando su desempeño.

El modelo dinámico expresa como responde el sistema a eventos externos, por tanto, la estructura de control del sistema principalmente se deriva del modelo dinámico.

Durante el diseño de objetos, el diseñador debe seguir los siguientes pasos:

- Combinar los tres modelos para obtener operaciones en clases.
- Diseñar algoritmos para implementar operaciones.
- Optimizar rutas de acceso a información.
- Implementar control para interacciones externas.
- Ajustar las estructuras de clases para aumentar herencia.
- Diseñar asociaciones.
- Determinar representación de objetos.
- Agrupar clases y asociaciones en módulos.

II.3. *Bases de Datos*

Por la versatilidad del paradigma Orientado a Objetos, no solamente permite desarrollar sistemas, sino que además puede ser utilizado para diseñar bases de datos; ya sean jerárquicas (modela los datos jerárquicamente), de red (modela los datos formando retículas), relacionales (basado en el concepto matemático de relación) u orientadas a objetos (se basa en los objetos que intervienen en una aplicación).

Los desarrolladores utilizan las bases de datos porque permite el acceso concurrente de varios usuarios, además de que existen actualmente múltiples sistemas manejador de bases de datos que permiten accesarlas. También debido a la seguridad, redundancia controlada, integridad y extensibilidad de la información.

Actualmente los enfoques más usados para diseñar bases de datos son dos:

- La primera toma en cuenta los atributos, es decir, se enlistan los atributos y se agrupan de tal manera que preserven sus dependencias. Esta manera de diseñar se conoce como el proceso de normalización en bases de datos relacionales.

- La segunda consiste en identificar entidades significativas a la aplicación y describirlas. Comunmente, el número de entidades es menor que el de atributos, haciendo el diseño por entidad más tratable.

El diseño de una base de datos debe comprender tres esquemas, que son el esquema externo, el conceptual y el interno. En el diseño van a existir tantos esquemas externos como aplicaciones existan, ya que se busca centralizar la información de todas las aplicaciones. En el esquema conceptual es donde se unifican los datos y se realiza un modelo de tablas ideal. El esquema interno trata con las limitaciones y aspectos del manejador de bases de datos (DBMS) específico.

Un sistema manejador de bases de datos es un programa computarizado diseñado para almacenar, consultar y controlar el acceso a una información permanente.

II.3.1. Bases de Datos Relacionales

Las bases de datos relacionales las inventó E. F. Codd y se basa en el concepto matemático de relación. Codd definió que un sistema manejador de bases de datos relacional (RDBMS) está compuesto por tres partes:

- Información que es presentada a través de relaciones.
- Operadores para manipular las relaciones.
- Reglas de Integridad con el fin de garantizar que los datos sean válidos.

Los sistemas manejadores de bases de datos almacenan a las relaciones en tablas. Las tablas tienen un número específico de columnas y un número arbitrario de renglones. Las columnas también reciben el nombre de atributos y los renglones se conocen también como entidades. Cada atributo debe tener asignado un dominio, es decir, los posibles valores que puede tomar un atributo.

Debido al concepto matemático de relaciones entre conjuntos en el que se basan las bases de datos relacionales, cuenta con las operaciones permitidas en los conjuntos como son: unión, intersección, diferencia y producto cartesiano.

La unión de dos relaciones es también una relación que contiene las entidades que se encuentran en una u otra relación. Suponiendo que a y b son relaciones con todos los dominios iguales, la unión entre éstas se denota como $a \cup b$.

La intersección es una relación que contiene aquellas entidades que se encuentren en ambas relaciones asociadas. La intersección se denota $a \cap b$ donde a y b son relaciones sobre los mismos dominios.

En la diferencia también es necesario que las relaciones involucradas operen sobre el mismo esquema conceptual o mismos dominios y atributos. Tomando en cuenta las mismas relaciones a y b, $(a\text{-}b)$ se define como la relación que contiene las entidades que estan en a pero no en b.

A diferencia de las operaciones anteriores, en el producto cartesiano es irrelevante el esquema conceptual en el que operan las relaciones. El producto cartesiano contiene todas las

entidades que resultan de concatenar cada entidad de la primera relación con otra de la segunda relación. Se denota como $a \times b$.

Además de las operaciones propias de los conjuntos, cuenta con operaciones específicas de las relaciones como permutación, proyección, selección y reunión.

La permutación se aplica a una sola relación y consiste en cambiar de orden los atributos. Se denota por $\Pi_{i\,j\,k}\,(a)$, donde los subíndices *i, j, k* indican el orden en que estarán los atributos de la relación original (*a*).

La proyección también es aplicada a una sola relación. Esta operación consiste en obtener una subrelación seleccionando algunos atributos. La proyección es denotada con $\pi\,(a)$, indicando los atributos seleccionados.

El resultado de la selección estará formada por aquellas entidades que cumplan con una condición sobre los valores de uno o más atributos. Es denotado por $\sigma_{\text{condición}}\,(a)$.

Finalmente la operación reunión (join) involucra a dos relaciones, se obtiene concatenando una entidad de la primera con otra de la segunda relación, de tal manera que cumpla con una condición en los dominios comunes. Se tratará de un producto cartesiano si no existen dominios comunes. Esta operación se denota con $a^{*}{}_{(\text{condición})}$.

En una tabla cada valor debe pertenecer al dominio de su atributo ó ser nulo. El valor va a ser nulo cuando se desconoce dicho valor.

Una llave primaria es la combinación de uno o más atributos cuyo valor hace de cada renglón único en una tabla. Una llave foránea es una llave primaria en una tabla a la que se hace referencia en otra.

En el modelo relacional existen dos reglas de integridad:

1. **Integridad por Entidad.-** Indica que cada tabla tiene exactamente una llave primaria y que cada componente no puede tener valor nulo. La llave primaria se refiere a la combinación de uno ó más atributos cuyo valor permite distinguir a cada renglón.

2. Integridad Referencial.- Se refiere a que cada llave foránea debe ser consistente a su llave primaria correspondiente. Una llave foránea hace referencia a una llave primaria en otra tabla.

II.3.2. Formas Normales

Existen reglas con el fin de evitar inconsistencias, llamadas formas normales. Existen varios niveles de forma normal. Para cada forma normal de mayor nivel, se agregan aspectos de la forma normal anterior.

- ***Primera Forma Normal (1FN)***

Una tabla está en primera forma normal cuando todos los valores de los atributos son atómicos.

- ***Segunda Forma Normal (2FN)***

Una tabla se encuentra en 2FN cuando satisface 1FN y cuando cada renglón tiene una llave primaria.

- ***Tercera Forma Normal (3FN)***

Una tabla en 3FN debe estar en 2FN y cuando no existe un atributo que no dependa directamente de la llave.

Cabe mencionar que existen otras formas normales de mayor nivel que generalmente no se necesitan, ya que una base de datos que cumpla con las tres primeras formas normales, se considera como un diseño aceptable. Al proceso de aplicar las formas normales a una base de datos recibe el nombre de normalización.

II.4. *Diseño Orientado a Objetos implementado en Bases de Datos Relacionales*

Gracias a la versatilidad del paradigma orientado a objetos, no solamente permite establecer las bases para diseñar un sistema y codificarlo, sino que también puede utilizarse para diseñar una base de datos relacional.

Para diseñar una base de datos, primero se realizan los pasos del análisis descritos anteriormente, construyendo un modelo de objetos por cada esquema externo que haya, es decir, para cada aplicación existente. Como el modelo de objetos ubica la estructura lógica de la información, facilita a los desarrolladores realizar un diseño de base de datos coherente, entendible, eficiente y correcto.

Posteriormente se traducen estos modelos de objetos en tablas ideales, para finalmente implementarse en el sistema manejador de bases de datos relacional que se haya elegido.

El modelo de objetos consiste en clases, asociaciones, generalizaciones y atributos. Y el objetivo del trabajo consiste en construir las tablas para el modelo de objetos.

A continuación se describen las reglas de mapeo necesarias para traducir los modelos de objetos a modelos de tablas.

II.4.1. Mapeo de Clases de Objetos a Tablas

Una clase se mapea en una o más tablas. Las instancias de una clase se pueden dividir horizontalmente y/o verticalmente dependiendo del criterio que se tome.

Por ejemplo si una clase tiene varias instancias y rara vez se hace referencia a algunas de ellas se puede hacer una partición horizontal aunque se debe saber en que tabla buscar. En cambio, si una clase tiene atributos con distintos patrones de acceso, convendría una partición vertical. La figura 2.11 muestra un ejemplo de estos tipos de partición.

En el modelo de tablas se enlista el identificador de la clase con sus atributos, añadiendo que campos pueden o no, ser nulos; y asignando su dominio correspondiente. También se especifica la llave primaria de cada tabla.

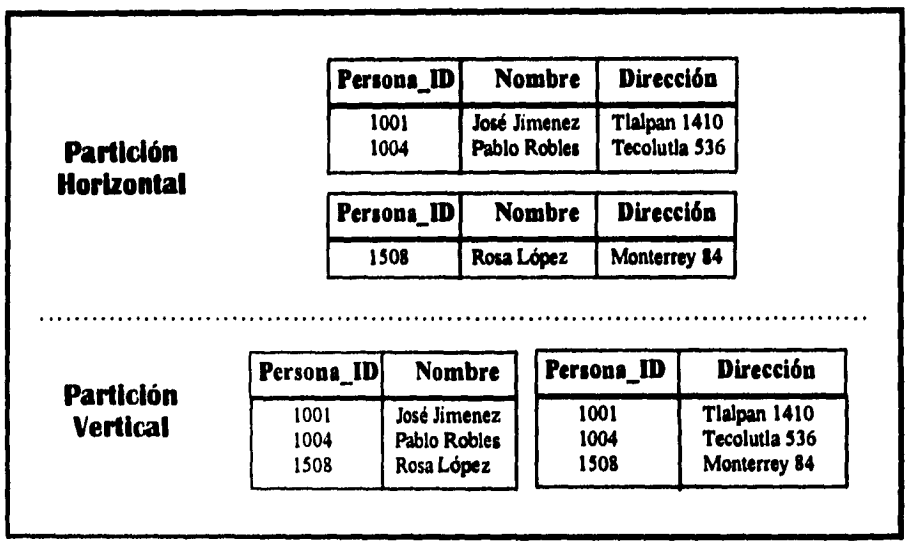


Figura 2.11

En la figura 2.12 se muestra la clase "Compañía" con los atributos nombre y dirección. La tabla que corresponde a esta clase, tiene los mismos atributos que el modelo de objetos (nombre, dirección), añadiendo un identificador (compañía-id), donde éste es la llave primaria a la tabla "Compañía". En la columna nulo? se indica si puede ser nulo o no.

II.4.2.Mapeo de Asociaciones Binarias a Tablas

Una asociación puede o no mapearse en una tabla, ya que depende de su multiplicidad y de la decisión del diseñador.

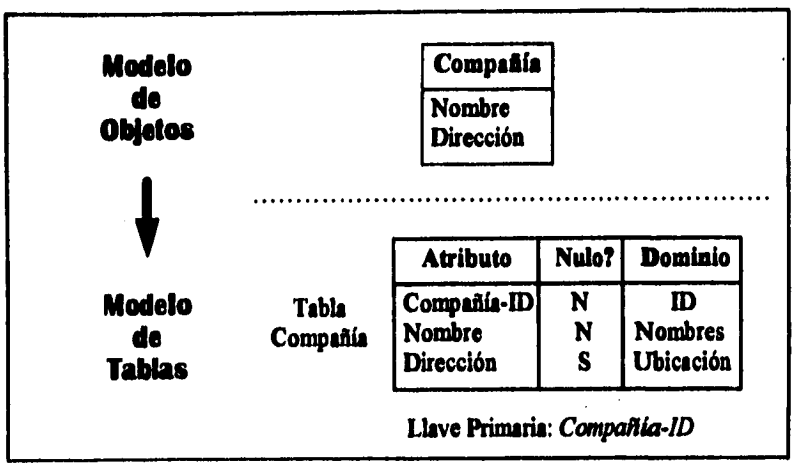


Figura 2.12

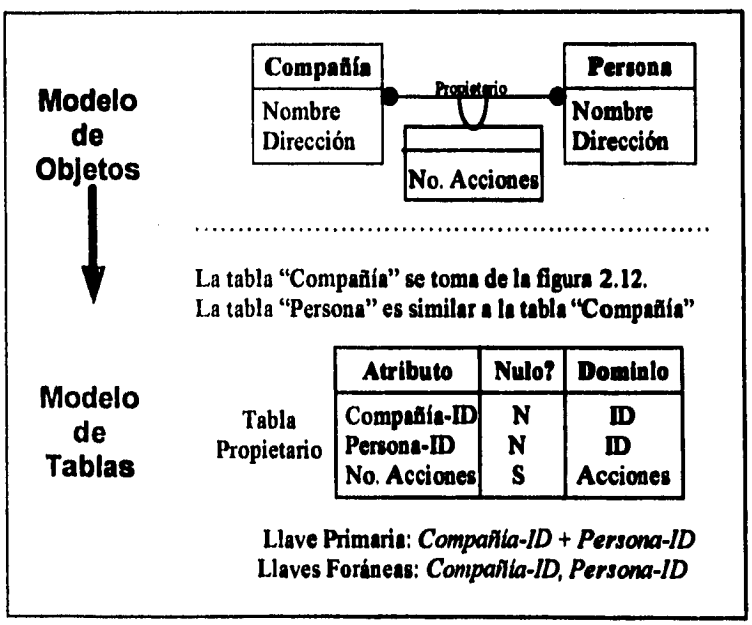


Figura 2.13

Una asociación de muchos a muchos siempre se mapea en una nueva tabla. Las llaves primarias de las dos clases relacionadas y cualquier liga de atributos serán atributos en la tabla asociada. Ambas llaves combinadas serán la posible llave primaria.

Por definición, una liga entre dos objetos requiere que ambos objetos sean conocidos, por lo que las llaves foráneas de una tabla asociada no pueden ser nulos.

En la figura 2.13 se muestra cómo se mapea una asociación de muchos-a-muchos.

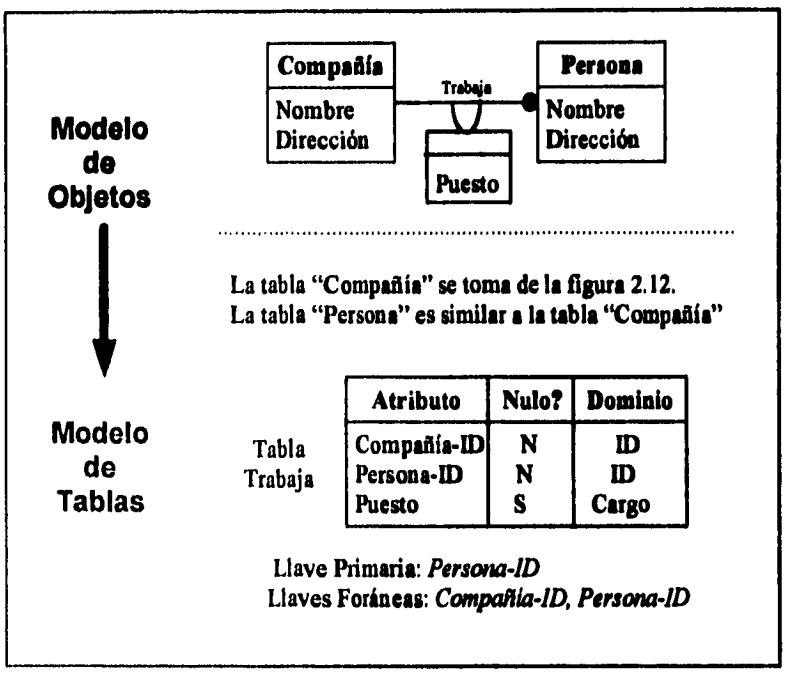


Figura 2.14

En las asociaciones uno-a-muchos (figura 2.14) y uno-a-uno hay dos maneras de mapear. La primera consiste en que se mapee en una tabla distinta aunque esto hace que la navegación sea más lenta. La segunda alternativa es aumentar una llave foránea en la tabla de multiplicidad muchos, en caso de tratarse de una asociación uno-a-muchos (figura 2.15). Si la asociación es uno-a-uno, se añade la llave foránea a cualquiera de las dos clases asociadas.

La tabla "Compañía" se toma de la figura 2.12.

	Atributo	Nulo?	Dominio
Tabla Persona	Persona-ID	N	ID
	Nombre	N	Nombres
	Dirección	S	Ubicación
	Compañía-ID	N	ID
	Puesto	S	Cargo

Llave Primaria: *Persona-ID*
Llave Foránea: *Compañía-ID*

Figura 2.15

II.4.3. Mapeo de Asociaciones Ternarias a Tablas

Las asociaciones ternarias se mapean en una tabla distinta cuyos atributos son las llaves de las clases asociadas y los atributos de liga, en caso de que existan, así como se muestra en la figura 2.16. Cabe mencionar que los roles son considerados también como atributos en la tabla de asociación. Además la llave primaria está formada por los identificadores de cada clase.

Un calificador se mapea en otra tabla que contiene al menos tres atributos (las llaves primarias de las clases relacionadas y el calificador).

En cuanto al mapeo de las agregaciones, por tratarse de un tipo de asociación, se sigue el mismo criterio.

II.4.4.Mapeo de Herencia a Tablas

Existen cuatro formas de mapear la herencia en tablas, tres de ellas se refieren a herencia sencilla y la cuarta manera de mapeo es para herencia múltiple, la cual, no es muy común que suceda.

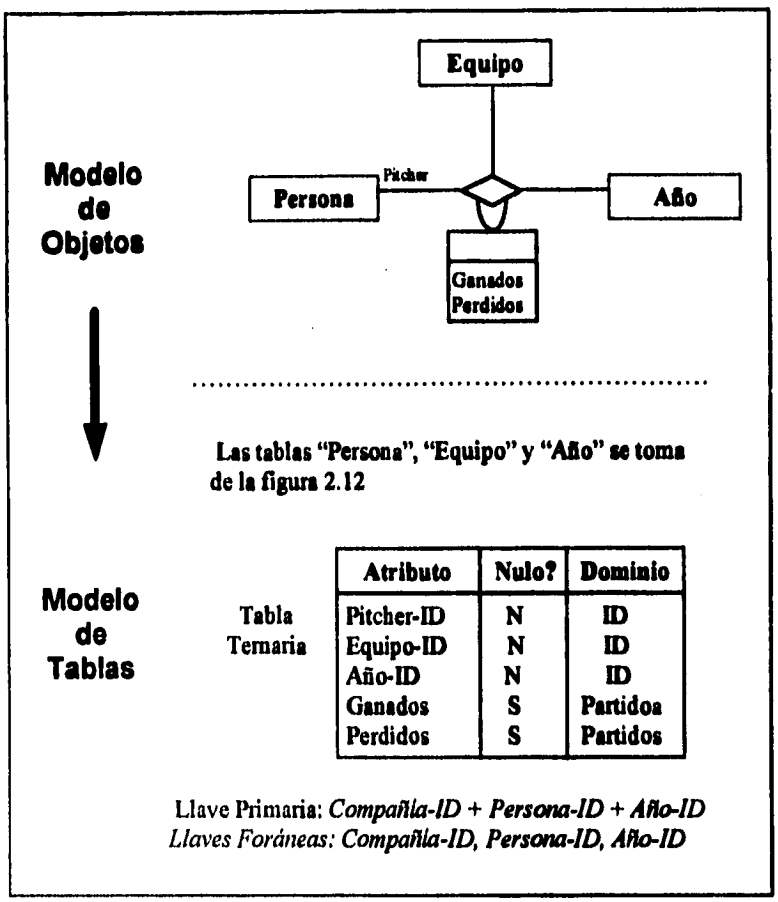


Figura 2.16

La primera consiste en mapear **la superclase y cada subclase a su vez en una tabla** distinta, donde cada subclase tiene como **llave foránea la llave de su superclase.**

La segunda manera es cuando **se decide no hacer una tabla de superclase, entonces** se ponen los atributos de ésta en cada tabla de las subclases. **Este tipo de mapeo involucra menos** tablas, agilizando la navegación. **Además, es recomendable cuando la superclase tiene pocos** atributos.

La tercera manera de mapeo para herencia sencilla es cuando se trasladan todos los atributos de las subclases al nivel de la superclase, disminuyendo aún más el número de tablas, aunque no cumple con la 3FN. Para poder identificar una ennada a que subclase pertenece es cuando tiene valores no nulos en los atributos de su subclase. Esto sucede cuando se optó por no hacer tablas de subclases.

La cuarta y última es cuando se trata de herencia múltiple aunque es poco común que se presente; las superclases y cada subclase se mapean en su tabla correspondiente añadiendo en cada tabla de las subclases como llaves foráneas a las llaves de sus superclases.

III. Desarrollo de una Aplicación siguiendo OMT

El objetivo de éste capítulo es dejar una idea más clara de la metodología OMT a través del desarrollo de una aplicación, abarcando las etapas del planteamiento del problema, su análisis dando como resultado el modelo de objetos, el dinámico y el funcional, continuando con la etapa de diseño donde se define la arquitectura del sistema, el diseño de objetos y el diseño de los datos de la aplicación; y finalmente se sigue con la implementación.

De esta manera permite ver de una forma práctica las decisiones que se necesitan ir tomando durante el desarrollo de una aplicación, haciendo resaltar que la aplicación misma y el desarrollador son los que marcan las pautas para tomar dichas decisiones.

III.1. *Planteamiento del Problema*

Como se ha visto cuando cualquier persona o empresa va almacenando información de interés personal es conveniente ordenarla para futuras consultas o simplemente para tener control sobre ella, es decir, con el fin de explotar dicha información.

Pues bien, esto mismo sucede cuando se trata de una colección de revistas, cualquiera que ésta sea. Considero que es importante ordenar la información de una revista, ya sea desde el punto de vista de la editorial como del lector.

Como editorial es importante porque se debe contar con un registro de todos los artículos que se han publicado para evitar repeticiones ó redundancias. Además, para tener conocimiento de qué aspectos se han escrito ya de un tema, ó una consulta rápida de localización de un artículo para añadir tópicos o actualizaciones del tema que se trató. En fin, puede ser de gran utilidad contar con un sistema que permita realizar esto.

Por otro lado, es conveniente para el lector contar con un sistema que le permita consultar la ubicación de un artículo sin tener que leer los índices de todas las revistas hasta encontrar el artículo que se busca, o los artículos del tema que se desea consultar. Esta tarea se complica y se convierte tediosa cuando el número de ejemplares con los que se cuenta es grande.

Así pues, el caso práctico del presente trabajo consiste en desarrollar un sistema que sea el índice de cualquier colección de revistas, bajo el paradigma orientado a objetos, utilizando la metodología OMT para su desarrollo.

A continuación se da una pequeña descripción formal del problema que se pretende analizar y resolver.

> La aplicación consiste en crear un sistema computarizado que sea el índice de cualquier colección de revistas. Además, un usuario debe poder usar las siguientes operaciones:

- Dar el índice de los artículos que contenga la revista 'X',
- Dado el nombre de un artículo, dar el número de revista,
- Dar los números de los ejemplares de revistas que contengan artículos que hablen de un tema en particular,
- Dar los nombres de los artículos escritos por el autor 'X',
- Dar de alta, de baja artículos o modificaciones de ellos.

III.2. *Etapa de Análisis*

La etapa de análisis se inicia con la descripción del problema, la cual no debe considerarse como inmutable. Una vez que se cuenta con ésta, se sigue con la construcción de los modelos que describen distintos aspectos del sistema como son los objetos que intervienen en él (modelo de objetos), su comportamiento dinámico (modelo dinámico) y funcional (modelo funcional).

Cabe mencionar que el resultado de la etapa de análisis son éstos tres modelos, sin descartar que la metodología OMT es un proceso iterativo, es decir, si en una etapa posterior se

añaden, modifican ó identifican nuevos aspectos de la aplicación, se puede regresar a la etapa que se necesite corregir sin que esto signifique un retroceso en el desarrollo de la aplicación.

III.2.1. Modelo de Objetos

Pues bien, el primer paso para la construcción del modelo de objetos es identificar las clases de objetos que intervienen en la aplicación. Para esto, se obtiene una lista de clases de objetos candidatos, la cual se constituye de sustantivos que se encuentran en la descripción del problema. En este nivel de análisis no es necesario preocuparse por herencia, ni multiplicidad, sino ubicar únicamente las clases correctas que posteriormente se organizarán.

Después de haber revisado los sustantivos en la descripción del problema, resulta la lista que se muestra en la figura 3.1. Cabe resaltar que debido al conocimiento que se tiene del mundo real y del problema mismo es válido añadir clases de objetos que no hayan sido consideradas ó contempladas en la descripción del problema.

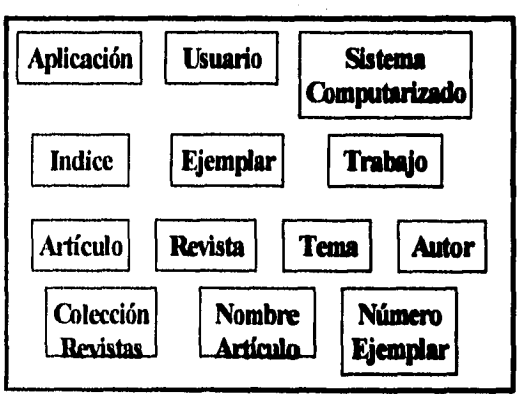


Figura 3.1

De ésta lista, se eliminan las clases que a juicio del analista considera que tienen poco que ver o nada con el problema, que expresan la misma información, que se pueden considerar atributos u operaciones de algunas clases, etc.

Por ejemplo de la lista de la figura 3.1, se suprimen:

- *"Aplicación"*, *"Trabajo"*, *"Sistema Computarizado"* por considerarse irrelevantes, ya que no tienen que ver con la aplicación en sí, más bien aparecieron debido a la redacción del caso práctico.
- Las clases redundantes, es decir, aquellas clases que expresan la misma información, y se mantiene aquella que tenga el nombre más descriptivo. Por ejemplo, *"Ejemplar"* se eliminó y *"Revista"* se guardó.
- *"Autor"*, *"Nombre Artículo"* y *"Tema"* se consideraron atributos por tratarse de nombres que describen al objeto *"Artículo"*, ya que un artículo consta de un nombre, habla de un tema en particular y fué elaborado por uno o varios autores. De la misma manera se eliminó *"Número Ejemplar"* por calificar a *"Revista"*.
- Por último, en lugar de *"Indice"* y *"Colección Revistas"*, se utilizará *"Biblioteca"*, porque de alguna manera, abarca la información de éstas.

La figura 3.2 muestra la lista resultante.

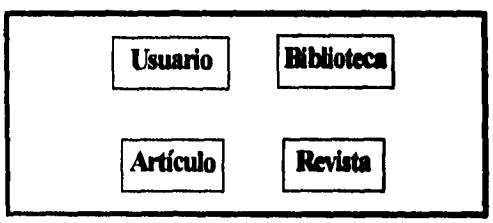


Figura 3.2

Una vez que se depuró la lista conformándola únicamente las clases de objetos correctas, se prepara un diccionario de datos que describa a cada clase, con el fin de evitar ambigüedades debido a que palabras aisladas se prestan fácilmente a tener varias interpretaciones (figura 3.3).

Ahora se busca identificar las asociaciones entre objetos, que como ya se había dicho es cualquier dependencia entre dos o más clases. También se considera una asociación cuando existe una referencia de un objeto a otro.

Al igual que en la identificación de las clases de objetos, se auxilia con la descripción del problema extrayendo una lista de las frases verbales que aparecen en ella, las cuales, normalmente corresponden a asociaciones (figura 3.4). De dicha lista, no se toman en cuenta aquellas frases que involucren clases eliminadas en el paso anterior, ni las asociaciones irrelevantes, que sean de implementación, ó aquellas que denoten acciones. A esta lista se le

agregan otras frases verbales que se consideren necesarias por el conocimiento que se tenga del dominio del problema.

Diccionario de Datos

Artículo
Escrito breve publicado en una revista, que contiene el nombre del escrito, el tema que trata y el/los autor(es).

Biblioteca
Conjunto de revistas que contiene artículos.

Revista
Publicación periódica, compuesta por varios artículos que tratan temas varios. Constituye un sólo ejemplar de una colección.

Usuario
Persona que desea obtener cierta información, o dar de alta, baja o modificar la colección de revistas, es decir, que pretende explotar la información de dicha colección.

Figura 3.3

Frases Verbales
Nombres de artículos que contiene una revista
De un artículo dar el número de revista
De un tema dar los números de revistas
Nombres de artículos escritos por un autor
Dar de alta artículos
Dar de baja artículos
Modificar artículos

Conocimiento del Dominio del Problema
Menú interactúa con el usuario
Menú interactúa con la biblioteca
Biblioteca está formada por revistas
Revista contiene varios artículos

Figura 3.4

A las asociaciones ternarias o de mayor grado, se deben descomponer en asociaciones binarias si es posible, aunque en la mayoría de éstas si se puede.

También se deben omitir asociaciones que puedan ser definidas a través de otras (asociaciones derivadas) ó bien aquellas definidas mediante condiciones.

Tanto como sea posible, se debe procurar que las clases de objetos, asociaciones y atributos expresen información por sí mismos.

De hecho, las asociaciones en ésta aplicación en particular, se obtuvieron a partir del conocimiento que se tiene sobre el mundo real y la aplicación, dando como resultado el diagrama de objetos preliminar que se muestra en la figura 3.5. Es preliminar, debido a que falta considerar los atributos y métodos de cada clase de objetos y refinar este modelo, utilizando herencia o jerarquía de clases.

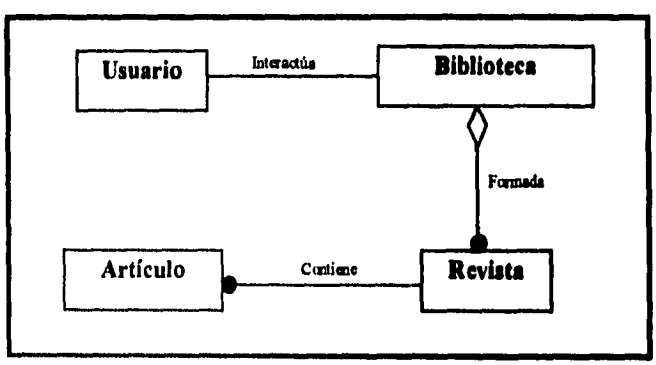


Figura 3.5

Lo que denota la figura 3.5 es que el usuario interactúa con la biblioteca, la cual está formada por varias revistas y por último una revista contiene varios artículos.

Cabe mencionar que las asociaciones se identificaron por el conocimiento del mundo real y no porque hayan sido frases verbales de la descripción del problema, se recomienda verificar, consultar o discutir con el cliente dichas asociaciones.

Posteriormente se busca identificar los atributos de las clases de objetos, algunos se obtuvieron de aquellos que se eliminaron como candidatos de clases precisamente por haberlos considerado como atributos.

A diferencia de los objetos y las asociaciones, es poco común que los atributos sean descritos en la descripción del problema y generalmente hay que hacer uso del conocimiento que se tiene del mundo real y de la misma aplicación para poder identificarlos. Únicamente hay que considerar aquellos atributos que estén directamente ligados a la aplicación.

Después se pueden añadir otros objetos más específicos con el fin de tener clases de objetos generales, como la clase "Menú", ya que un usuario final, normalmente no tiene acceso directo a la biblioteca, sino que cuenta con una interfaz amigable para manipular la información, y en este caso, esta interfaz está dada por la clase "Menú". Por tanto, esta se puede definir como:

> ***Menú*** es el objeto que interactúa con el usuario y la biblioteca para que se realice alguna acción.

Así pues, en el problema del índice de una colección de revistas, las operaciones de los objetos, la biblioteca es la que se va a encargar de dar artículos dado el número de revista, dar artículos según el autor, dar número de revista de un artículo, a partir de un tema dar los números de revistas que contenga artículos que hablen de dicho tema y las operaciones clásicas de una biblioteca, como son alta, baja y modificar. Por último, Menú, es quien se va a encargar de preguntar los datos necesarios para que la biblioteca realice las búsquedas deseadas. Dando como resultado el diagrama de objetos que se muestra en la figura 3.6.

El siguiente paso es organizar las clases utilizando herencia. Se puede aplicar la herencia en dos direcciones, una generalizando aspectos comunes de clases existentes en una superclase y la otra es refinando clases ya existentes pero en una subclase.

No se descarta la idea de utilizar herencia múltiple, aunque esto complica tanto la idea conceptual del sistema como su implementación.

En el caso práctico no se utilizó herencia, basándose en el criterio descrito anteriormente.

Por tal razón el modelo de objetos de la aplicación índice, estaría representado por la figura 3.6.

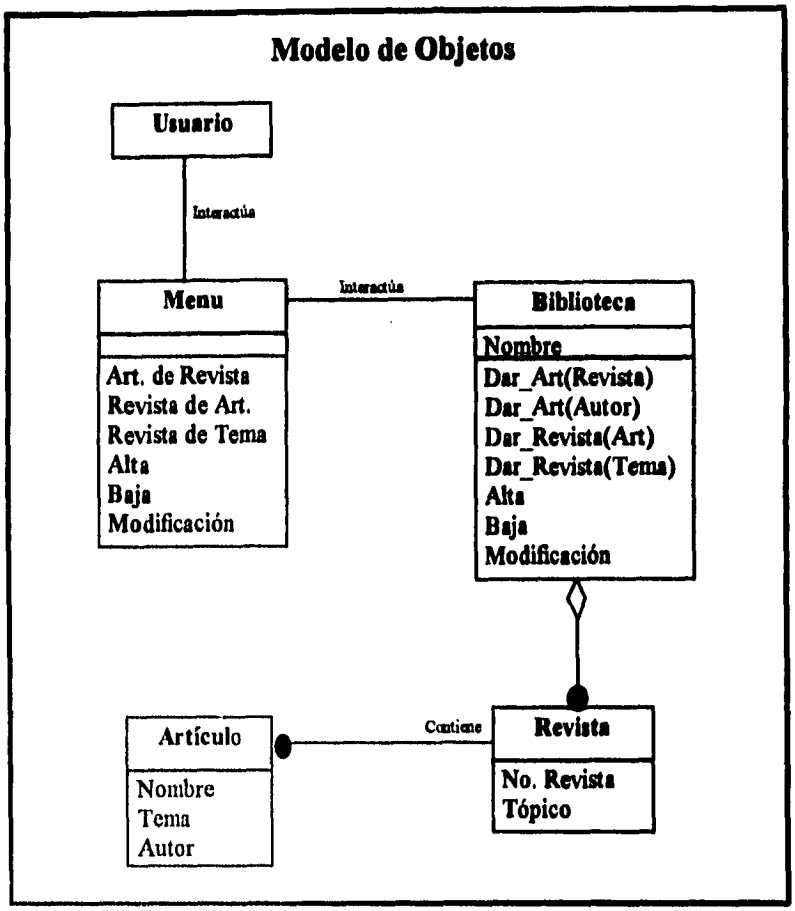


Figura 3.6

III.2.2. Modelo Dinámico

El siguiente paso de la etapa de análisis, es el desarrollo del Modelo Dinámico con el fin de que refleje el comportamiento de un sistema y los objetos dependiendo del tiempo. Esto es, los eventos (impulsos) y estados (respuestas) de aquellas clases del Modelo de Objetos que tengan una participación importante en el comportamiento dinámico del sistema.

El modelo dinámico es importante en sistemas interactivos, en cambio en sistemas puramente estáticos como en aquellos que se depositan datos su comportamiento dinámico es casi nulo.

El modelo dinámico está constituido por un conjunto de diagramas de estados. Únicamente a los objetos con comportamiento dinámico significativo se les desarrolla un diagrama de estados para que muestren los eventos que reciben y mandan.

Para facilitar la creación de éste modelo, es conveniente crear varios escenarios, los cuales como ya se había dicho, son secuencias de eventos que ocurren cuando hay cambio de información entre objetos. Estos escenarios permiten identificar los eventos entre objetos.

Primero se crean escenarios típicos entre los usuarios y el sistema sin entradas inusuales o casos de error. Después se consideran casos especiales, como entradas repetidas o inválidas.

Escenario para obtener el índice de una Revista.

Usuario escoge la opción de obtener artículos de una revista.
El menú pide número de revista de la que desea el índice.
Usuario da número de revista.
El menú pide artículos de la revista deseada.
La biblioteca da los artículos de la revista.

Escenario con error

Usuario escoge la opción de saber las revistas que hablen de un tema específico.
El menú pide el tema.
Usuario da el tema.
El menú pide las revistas que contengan artículos del tema deseado.
La biblioteca informa que el tema no se ha tratado.

Escenario para modificar un artículo

Usuario escoge la opción de modificación.
El menú pide los datos del artículo que desea modificar.
Usuario proporciona los datos (nombre del artículo, número de revista, tema que trata).
El menú pide los nuevos datos del artículo.
Usuario proporciona nuevos datos.
El menú pide que se modifique.

La biblioteca lo modifica.

De cada escenario se deben identificar los eventos. Se consideran eventos, todo tipo de señales, entradas, decisiones, interrupciones y acciones de usuarios o para ellos. También se considera evento a una acción que ejecuta un objeto transmitiendo información.

Así pues, para cada escenario se realiza una traza para identificar los eventos fácilmente. Una traza de eventos es un diagrama que muestra el emisor y receptor de eventos y la secuencia de estos. Este diagrama muestra cada objeto como una línea vertical y cada evento como una flecha horizontal con dirección del objeto emisor al receptor. El tiempo se va incrementando de arriba hacia abajo, aunque la distancia es irrelevante ya que no muestra el tiempo real entre un evento y otro.

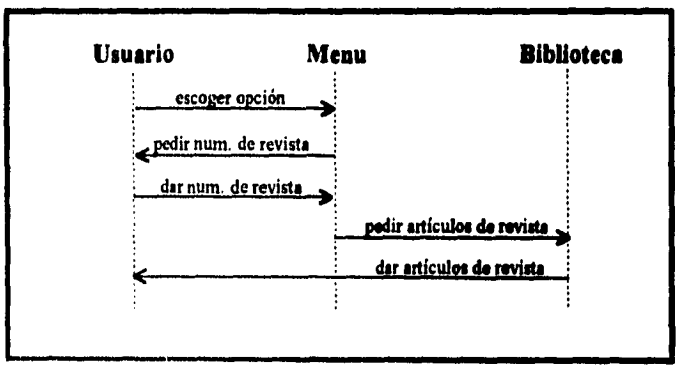


Figura 3.7

Por ejemplo, la figura 3.7 representa la traza del escenario para obtener el índice de una revista. De aquí, se consideran los eventos que afectan a un solo objeto. El intervalo entre un evento y otro es un estado del objeto que se está analizando.

Una vez que ya se identificaron los eventos y los estados, se crea un diagrama de estados, el cual, en este momento representa un camino del modelo dinámico. Esta operación se repite para cada escenario que se creó, atacando a cada nueva secuencia de eventos como un camino alternativo de un estado ya existente. De esta manera se va conformando el modelo dinámico definitivo.

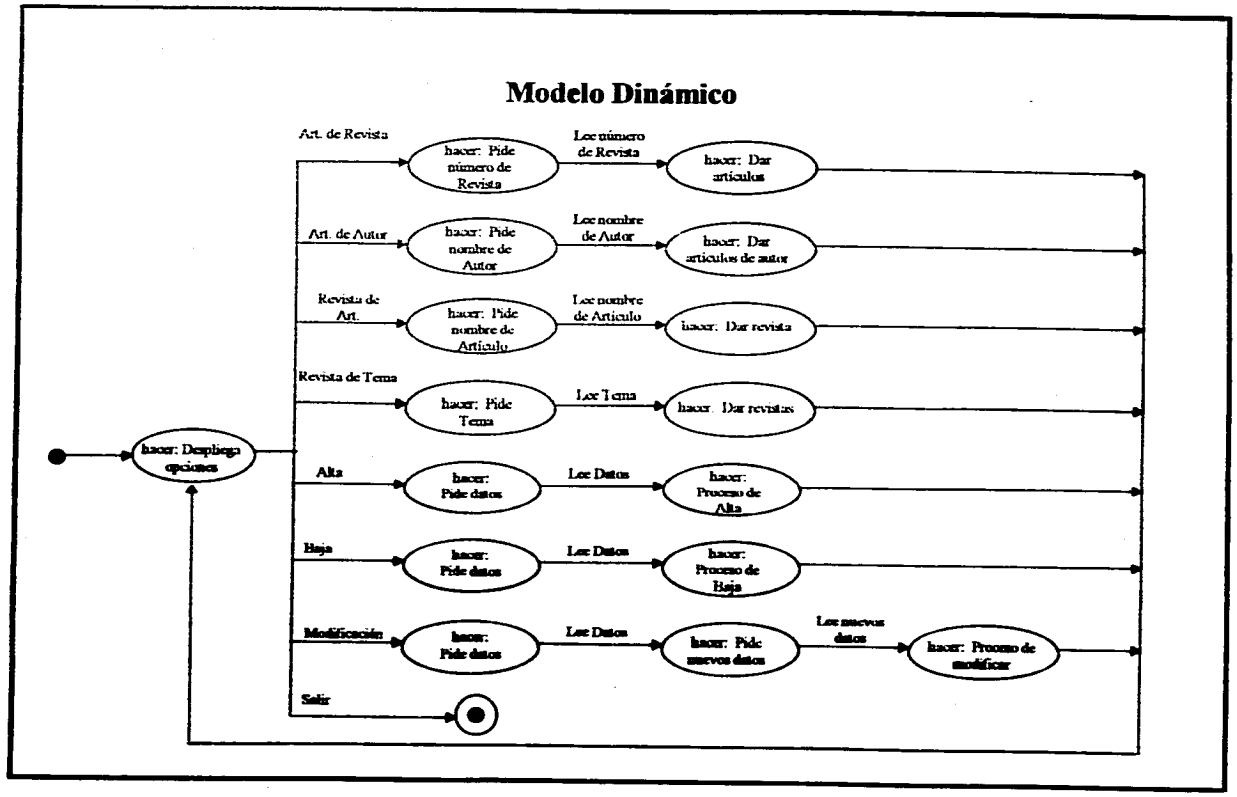

Modelo Dinámico
Art. de Revista
hacer: Pide número de Revista
Lee número de Revista
hacer: Dar artículos
Art. de Autor
hacer: Pide nombre de Autor
Lee nombre de Autor
hacer: Dar artículos de autor
Revista de Art.
hacer: Pide nombre de Artículo
Lee nombre de Artículo
hacer: Dar revista
Revista de Tema
hacer: Pide Tema
Lee Tema
hacer: Dar revistas
hacer: Despliega opciones
Alta
hacer: Pide datos
Lee Datos
hacer: Proceso de Alta
Baja
hacer: Pide datos
Lee Datos
hacer: Proceso de Baja
Modificación
hacer: Pide datos
Lee Datos
hacer: Pide nuevos datos
Lee nuevos datos
hacer: Proceso de modificar
Salir

Figura 3.8

Este proceso se concentra únicamente en aquellas clases con iteraciones importantes en la aplicación.

En el caso práctico índice, la clase menú es la única clase a la que se le desarrolló diagrama de estados por las razones anteriormente citadas. Por lo tanto, la figura 3.8, muestra el modelo dinámico de la aplicación, cuyo estado inicial es cuando se despliegan las opciones que el usuario tiene ya sea para realizar una consulta o actualización.

Por ejemplo, haciendo referencia a la figura 3.8, para llegar al estado donde se pide el número de revista, el usuario debió escoger la opción de artículos de una revista (representada con la flecha que dice Art. de Revista) para llegar al estado donde se dan los artículos regresando nuevamente al estado inicial.

Cabe mencionar que para facilitar aún más el desarrollo del modelo dinámico se elaboraron más escenarios con sus respectivas trazas, los cuales no se discutirán debido a que se trabajaron de manera semejante.

Como se puede ver, el modelo dinámico muestra los distintos caminos que pueden tomar los objetos cuando corre la aplicación, mostrando sus distintos estados y los eventos que deben suceder para llegar a ellos.

III.2.3.Modelo Funcional

El siguiente paso es la construcción del modelo funcional, para mostrar como se calculan los datos, sin tomar en cuenta donde ni porque se llevan a cabo. También muestra las relaciones entre los datos, es decir, que datos dependen de que datos y las funciones que los relacionan. La representación del modelo funcional es a través de un conjunto de diagramas de flujos de datos.

Los procesos en los diagramas de flujos de datos corresponden a acciones o actividades del modelo dinámico, por tal razón es mejor construir el modelo funcional después de éste.

Para construir los diagramas de flujos de datos, es necesario identificar los datos de salida y de entrada para cada proceso, mostrando como cada dato de salida es calculado a partir de datos de entrada. Identificando también los objetos actores y pasivos.

Una vez que se hayan construido los diagramas de flujos de datos, se deben describir cada función. La descripciones se pueden expresar de distintas maneras, ya sea en pseudocódigo, lenguaje natural, lenguaje matemático o en tablas de decisión.

En las figuras 3.9 y 3.10 se puede observar el diagrama de flujos de datos para obtener los nombres de los artículos que pertenecen a una revista específica y su respectiva descripción.

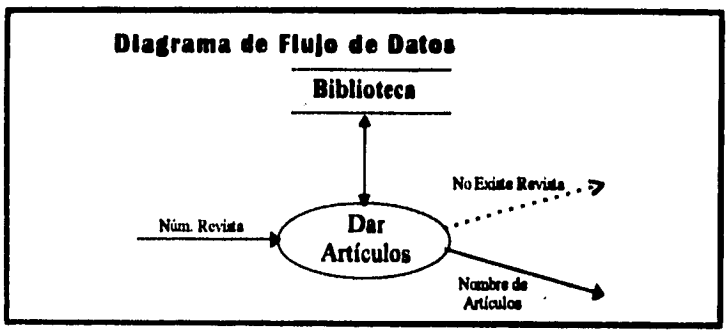


Figura 3.9

Descripción

Dar Articulos (Num. Revista) ---> mensaje, nombres de articulos
Si Existe Num. Revista,
desplegar nombres de articulos
si no,
mensaje

Figura 3.10

El diagrama de flujo de datos para obtener los nombres de artículos escritos por un autor, con su descripción se ilustra en la figura 3.11.

En la figura 3.12 se muestra el diagrama de flujo de datos para el proceso de saber el número de la revista a la que pertenece un artículo, con su respectiva descripción.

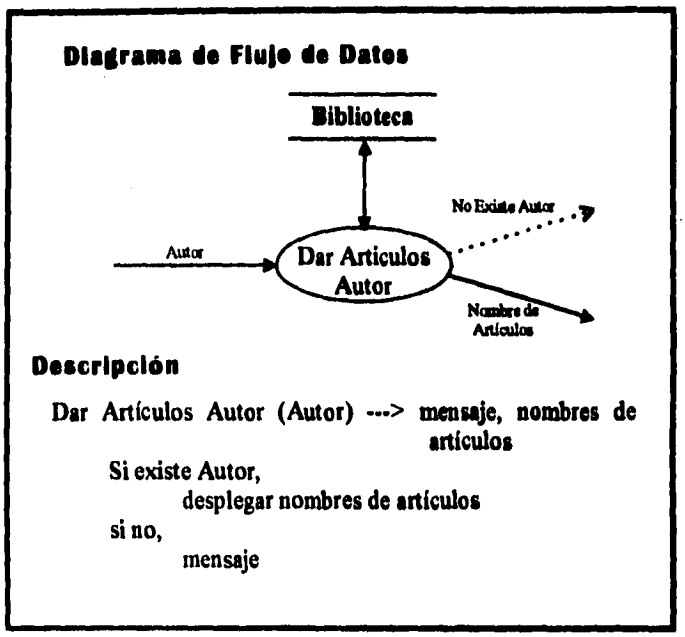


Figura 3.11

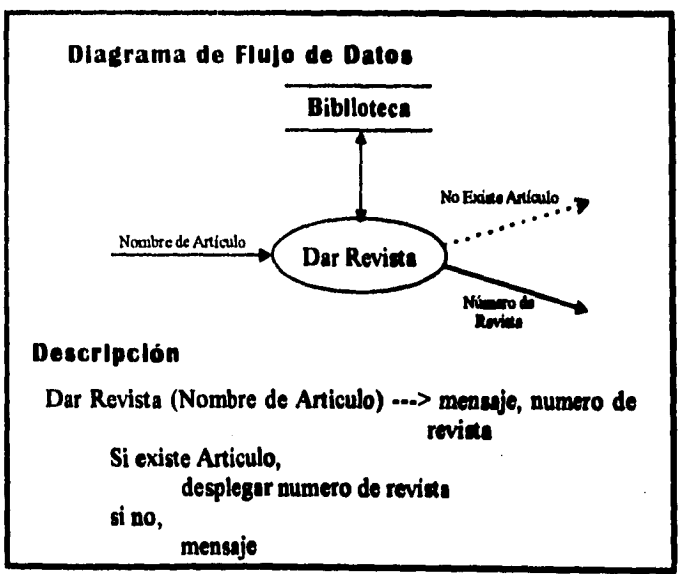


Figura 3.12

El diagrama de flujo de datos para obtener la lista de números de revistas que contenga(n) artículo(s) que traten un tema en particular está ilustrado en la figura 3.13, y su respectiva descripción se muestra en la figura 3.14.

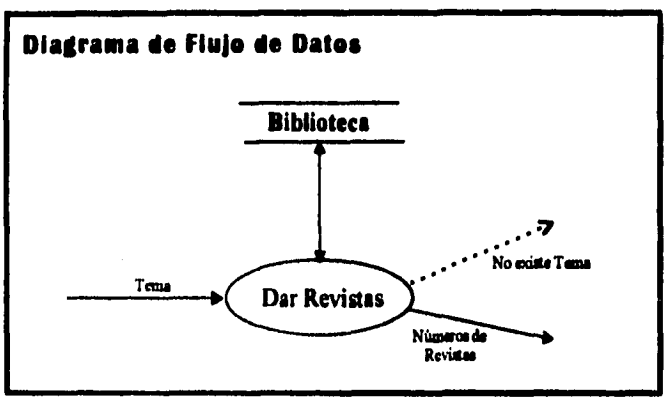


Figura 3.13

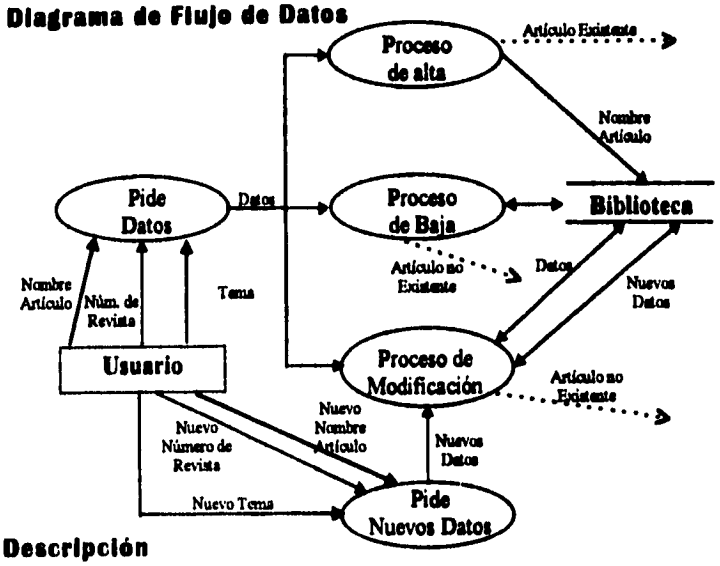
Descripción

Dar Revistas (Tema) ---> **Mensaje, Numerosde Revistas**
Si existe Tema,
Desplegar Numeros de Revistas
si no,
Mensaje

Figura 3.14

Finalmente en la figura 3.15 se muestra el diagrama de flujo de datos y la descripción de cualquier proceso que involucre una actualización respectivamente, esto es, para dar de baja o de alta un artículo o también si se trata de una modificación de datos (nombre de artículo, tema o ejemplar).

Diagrama de Flujo de Datos



Descripción

Pide Datos (Nombre Articulo, Num. de Revista, Tema)---> Datos
Si Nombre Articulo, Num. de Revista y Tema son correctos, entonces
Datos
Si no, Mensaje

Pide Nuevos Datos (Nuevo Nombre Articulo, Nuevo Num. de Revista, Nuevo Tema)---> Nuevos Datos
Si Nuevo Nombre Articulo, Nuevo Num. de Revista, Nuevo Tema son correctos, entonces
Nuevos Datos
Si no, Mensaje

Proceso de alta (Datos) ---> Mensaje , Datos
Si Existe Datos en Biblioteca, entonces
Mensaje
si no, Añadir Datos

Proceso de baja (Datos) ---> Mensaje , Datos
Si Existe Datos en Biblioteca, entonces
Borrar Datos en Biblioteca
si no, Mensaje

Proceso de Modificación (Datos, Nuevos Datos) ---> Mensaje , Datos , Nuevos Datos
Si Existe Datos en Biblioteca, entonces
Borra Datos de Biblioteca
Añade Nuevos Datos en Biblioteca
si no, Mensaje

Figura 3.15

Cabe mencionar que el conjunto de diagramas de flujos de estados (ilustrados en las figuras 3.9, 3.11, 3.12, 3.13 y 3.15) conforman el modelo funcional de la aplicación Indice.

III.3. *Etapa de Diseño*

Una vez que se realizó el análisis, se cuenta ya con tres modelos que representan los tres principales aspectos del sistema, estos son, los objetos y sus relaciones entre sí, el flujo dinámico de control y las transformaciones funcionales de datos.

Después se sigue con la etapa de diseño que consta de dos partes: diseño del sistema y el diseño de objetos

En el diseño del sistema se busca definir la arquitectura de este, además se toman decisiones de alto nivel como decidir entre espacio y tiempo, hardware y software, etc. También, en esta etapa se divide el sistema en subsistemas y se designa cada uno a unidades de hardware o software. Además se decide la implementación de la información, ya sea a través de archivos, estructuras de datos, o bien, bases de datos.

Durante el diseño de objetos se elaboran y refinan los modelos que se elaboraron en la etapa de análisis para producir un diseño práctico. En esta etapa, se hace énfasis a conceptos computacionales, se determina las definiciones completas de clases y asociaciones que se usarán en la implementación, así como también las interfaces y algoritmos de métodos que se usarán para implementar las operaciones. En el diseño de objetos se añaden clases intermedias de implementación y se optimizan las estructuras de datos y algoritmos.

Por tanto, el diseño de objetos es análogo a la etapa de diseño del ciclo de vida clásico de desarrollo de software.

III.3.1. Diseño del Sistema

La arquitectura del sistema Indice es híbrida de una interfaz interactiva y manejador de transacciones. El menú es una interfaz interactiva ya que como se definió anteriormente, es un objeto que interactúa con el usuario para que le proporcione los datos necesarios para realizar una transacción (consulta ó actualización). La biblioteca tiene un manejador de transacciones pues mantiene y actualiza una base de datos, donde se encuentra almacenada toda la información de los artículos y revistas.

La figura 3.16 muestra la arquitectura del sistema Indice. Por tratarse de un sistema pequeño, existe un único subsistema que es el de la biblioteca. La topología del sistema está constituida por la interfaz del usuario y la interfaz de opciones.

En el único lugar donde se almacena, y actualiza la información es en la biblioteca.

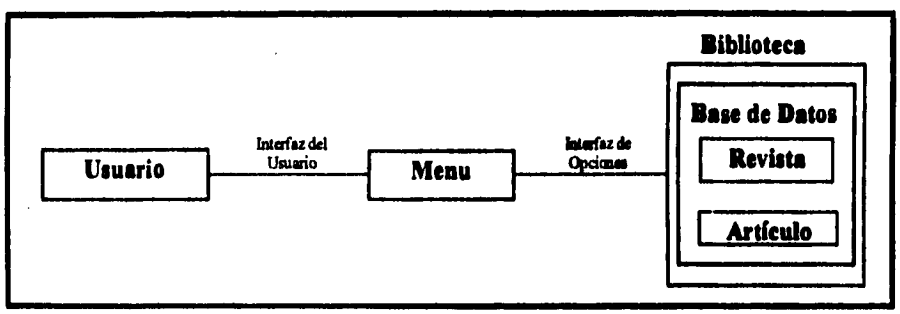


Figura 3.16

III.3.2.Diseño de Objetos

Los objetos que se identificaron en el análisis sirven como esqueleto para el diseño de objetos aunque el diseñador debe escoger entre varias alternativas de implementación, tratando de minimizar tiempo de ejecución, memoria y costos.

Por lo pequeña que es la aplicación y por tratarse básicamente de un problema de consulta y mantención de cierta información, el paradigma Orientado a Objetos y la metodología OMT, permite utilizar el modelo de objetos que se obtuvo en el análisis como una herramienta para realizar un diseño de base de datos. Lo importante de éste concepto, es que permite a los desarrolladores de sistemas, llevar a cabo un análisis Orientado a Objetos con todas las ventajas que éste ofrece y diseñar una base de datos relacional, lo que permite contar con una extensa gama de manejadores de bases de datos relacionales, que cuentan ya con cierta madurez.

Así pues, el modelo de objetos de la figura 3.6, las clases de objetos de *"Revista"* y *"Artículo"* son las que contienen información. Por tal razón, son las que se van a mapear en tablas.

Una clase se mapea en una o más tablas, aunque también, una tabla puede corresponder a más de una clase.

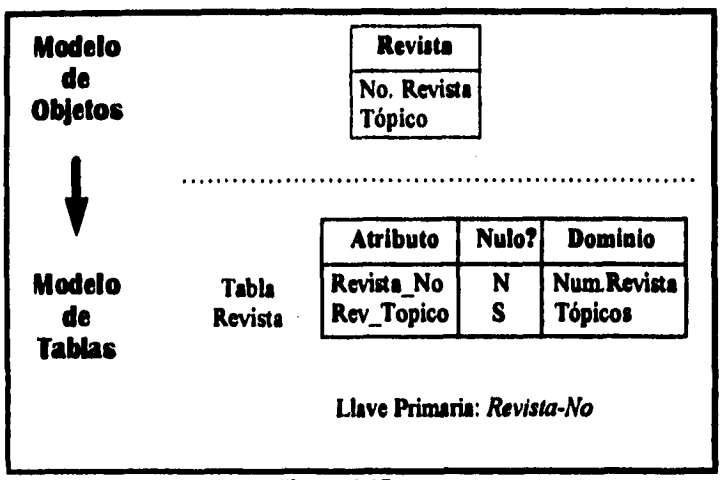


Figura 3.17

La figura 3.17 muestra como se mapea la clase "*Revista*" que tiene los atributos *"No. Revista" y "Tópico"*. Aunque generalmente para mapear una clase de objetos en una tabla se añade un identificador, además de sus atributos para que constituya la llave primaria, en este caso, no es necesario añadirlo ya que el número de revista tal cual, representa ya un identificador. Cabe mencionar que no puede ser nulo por ser llave primaria.

El mapeo de la clase "*Artículo*" es similar a la clase "*Revista*", ya que los atributos de la tabla artículo van a ser los mismos que la clase (nombre, tema) aumentando un identificador que constituye la llave primaria. Aquí el nombre de artículo no puede ser nulo, aunque el tema y el autor si pueden ser nulo (figura 3.18).

Existen dos maneras para mapear las asociaciones binarias con multiplicidad uno-a-muchos, el cual, es el caso de la asociación "*Contiene*" del modelo de objetos de la figura 3.6. La primera es mapearla en una tabla distinta, pero no es lo óptimo, ya que provoca una navegación más lenta, pues para realizar una consulta que involucre esta asociación, tiene una tabla más donde llevar a cabo una búsqueda. La segunda alternativa es aumentar una llave foránea en la tabla que corresponde a la clase que tiene multiplicidad de muchos. La figura 3.19 muestra como se modifica la tabla artículo para mapear la asociación "Contiene", donde el atributo "Revista_ID" no puede ser nulo ya que cualquier artículo pertenece a una revista (Esto depende de la aplicación ya que, si se trata de una que controla la publicación de artículos en una

editorial, pueden existir artículos que aún no estén publicados o asignados a una revista en particular).

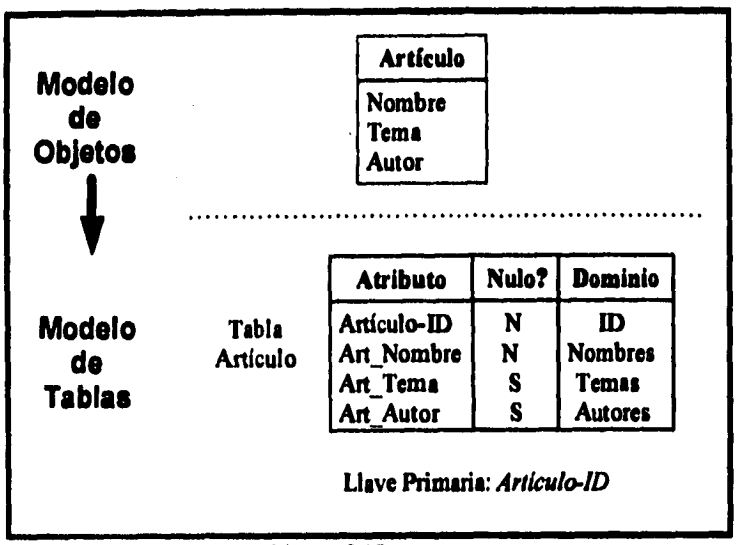


Figura 3.18

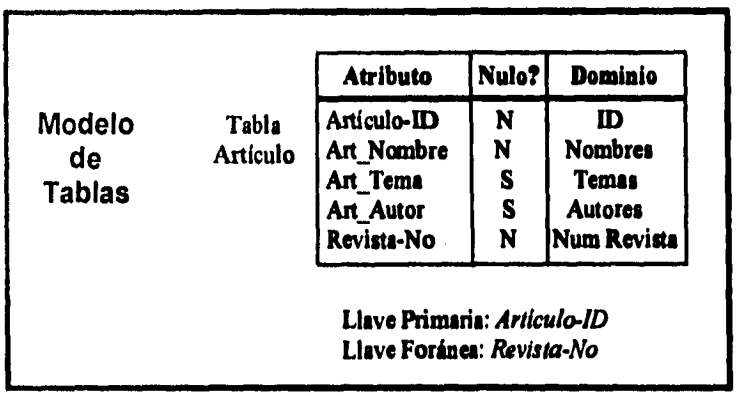


Figura 3.19

Con esto, se da por terminado la parte de diseño que concierne a la base de datos de la aplicación Indice, que son las figuras 3.17 y 3.19.

Ahora tomando en cuenta la estructura de la base de datos, se muestra en la figura 3.20 el algoritmo del proceso "Dar_Articulos", que sirve para consultar artículos de un ejemplar. Dicho algoritmo consiste en seleccionar el atributo "Art_Nombre", consultando la tabla de "Revista" y la tabla "Artículo" de aquellos renglones de la base de datos que satisfagan que el atributo "Revista_No" sea igual al número de revista que desee el usuario. De maners similar son los algoritmos de los procesos "Dar_Articulos_Autor" que sirve para consultar los artículos escritos por un autor, "Dar_Revista" que da la revista en que está un artículo consultado por nombre y finalmente "Dar_Revistas" que proporciona el número de revistas y nombres de artículos que trstan de un tema en particular. Los algoritmos de estos procesos se ilustran en las figuras 3.21, 3.22 y 3.23 respectivamente.

```
Dar_Articulos (Num_Revista) ---> Tabla
        Seleccionar  Art_Nombre
                De  Tabla_Revista, Tabla_Artículo
             Donde  Revista_No = Num_Revista
```

Figura 3.20

```
Dar_Articulos_Autor (Autor) ---> Tabla
        Seleccionar  Art_Nombre, Revista_No
                De  Tabla_Revista, Tabla_Artículo
             Donde  Art_Autor = Autor
```

Figura 3.21

```
Dar_Revista (Nombre_Artículo) ---> Revista
        Seleccionar  Revista_No
                De  Tabla_Revista, Tabla_Artículo
             Donde  Art_Nombre = Nombre_Artículo
```

Figura 3.22

```
Dar Revistas (Tema) ---> Tabla
    Seleccionar Revista_No, Art_Nombre
        De Tabla_Revista, Tabla_Artículo
    Donde Art_Tema = Tema
```

Figura 3.23

En cuanto a los procesos de actualización que son alta, baja y modificación, la descripción dada en la etapa de análisis (figura 3.15) constituye el algoritmo de dichos procesos.

III.4. *Implementación*

La implementación es la etapa que sigue al diseño, que consiste en traducir los algoritmos de la etapa anterior en lenguaje de programación, con el fin de darle al usuario la aplicación que resuelve su problema.

Aunque las decisiones importantes del sistema se tomaron en la etapa de diseño, no se le debe restar importancia a la escritura de código ya que eso puede facilitar o complicar ajustes o modificaciones futuras.

Con lo que respecta a la aplicación **Indice, para la creación de la base de datos, en** un principio se tomó en cuenta la gran diversidad de sistemas manejadores de bases de datos relacionales y se decidió utilizar ACCESS de Microsoft por su interfaz amigable para la creación de tablas y establecer las relaciones entre éstas, facilitando esta tarea, pero esto originó que se incrementaban los requerimientos tanto de desarrollo como de ejecución. Por tal motivo, se decidió finalmente utilizar el Data Manager con el que cuenta Visual Basic, para la creación de la base de datos, que finalmente crea también bases de datos con el formato de ACCESS pero de versiones anteriores (2.0 y 2.1).

En cuanto a la interfaz interactiva entre el usuario y la base de datos, se utilizó Visual Basic por tratarse de un ambiente que facilita grandemente la creación de interfaces en aplicaciones Windows, además haciendo casi transparente, el manejo de información en bases de datos creadas en el Data Manager. La ventaja principal que ofrece este paquete, es que el programador no tiene que escribir mucho código para crear una interfaz amigable a los ojos del usuario, sino que le presta mayor atención en llevar a cabo lo que se espera que haga el sistema.

Para instalar la aplicación Indice en el disco duro de la máquina se siguen las instrucciones desde DOS; (ya que los archivos en el diskette adjunto estan empaquetados)

1. **Insertar el diskette en una unidad de disco,**
2. **Ejecutar la siguiente instrucción desde la unidad de disco donde se encuentre el diskette:** *a:\> instalar -d c:*

De esta manera se creará un directorio en c: llamado Indice, el cual contiene todos los archivos necesarios para ejecutar la aplicación. Cabe mencionar que requiere de Windows para correr.

La implementación se inició con la creación de las tablas que conforman la base de datos. La figura 3.24 ilustra las tablas *Revista* y *Articulo* que se relacionan mediante el atributo '*rev_numero*', conformando de esta manera la base de datos resultante de la aplicación Indice.

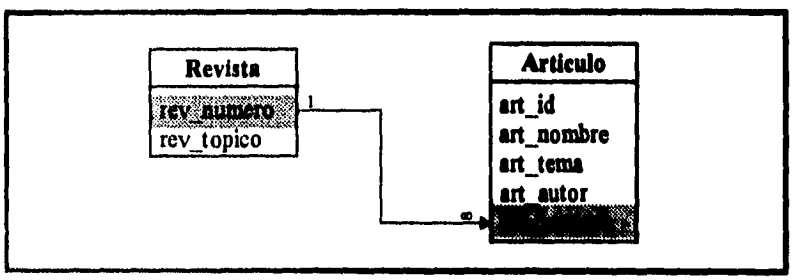


Figura 3.24

Una aplicación desarrollada en Visual Basic es un conjunto de formas, es decir, cuando la aplicación ya es ejecutable estas formas son ventanas que van apareciendo según se indiquen. Es por eso, que en la implementación de la aplicación Indice, se realizó primero la pantalla principal, la cual, consta de una serie de opciones de consulta, un menú para realizar actualizaciones a la base de datos y los clásicos botones de 'OK' y 'Cancelar' que tienen todas las aplicaciones que corren en ambiente Windows. Cada elemento de las formas a su vez tienen asignados varios eventos, por ejemplo, a los botones se les puede hacer clic o doble clic o arrastrar el mouse. Para cada evento se le puede o no asignar una serie de instrucciones que ejecute cuando este ocurra. De esta manera a grandes rasgos es como se desarrollan las aplicaciones con Visual Basic.

Con lo que respecta a la aplicación **Indice** ya corriendo, cuenta con una pantalla principal de opciones que se muestra en la figura 3.25, donde el usuario puede seleccionar la consulta que desee realizar. Esta ventana cuenta también con un menú con opciones para llevar a cabo una actualización a la base de datos o simplemente terminar o cerrar la aplicación.

Existen dos maneras de llevar a cabo alguna consulta, la primera es haciendo clic con el mouse a la opción que desea realizar y posteriormente 'Aceptar'. La segunda forma es únicamente haciendo doble clic a la consulta deseada.

Una vez que se haya seleccionado la consulta deseada aparecerá una pantalla asociada donde podrá especificar los datos necesarios para llevarla a cabo, posteriormente debe hacer clic al botón 'Aceptar' y finalmente se desplegarán los resultados de dicha consulta.

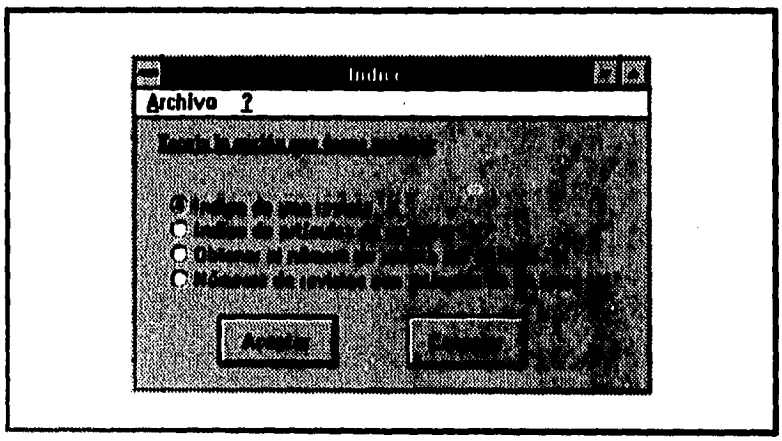


Figura 3.25

Al igual que todas las aplicaciones Windows cuenta con el botón 'Cancelar', cuya función no varía. Al ejecutar este botón, automáticamente se regresará a la pantalla principal.

Las actualizaciones se llevarán a cabo mediante tablas, es decir, deberá seleccionar la acción 'Actualización' del menú Archivo y escoger la tabla (Revista o Artículo) a la que se desea darle mantenimiento. (figura 3.26)

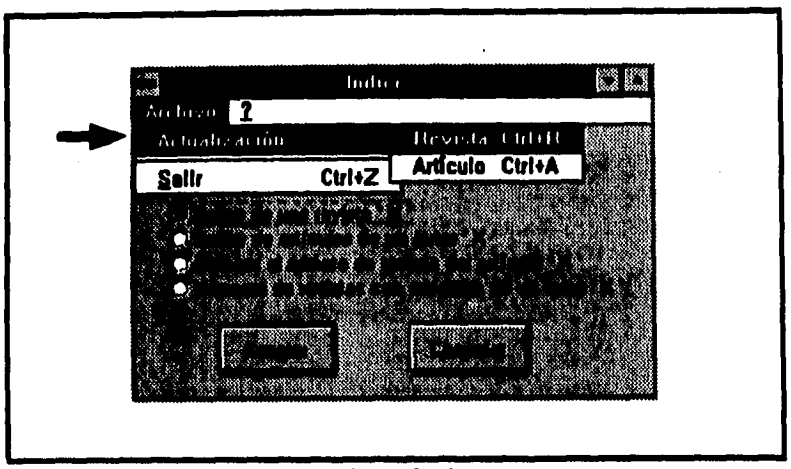


Figura 3.26

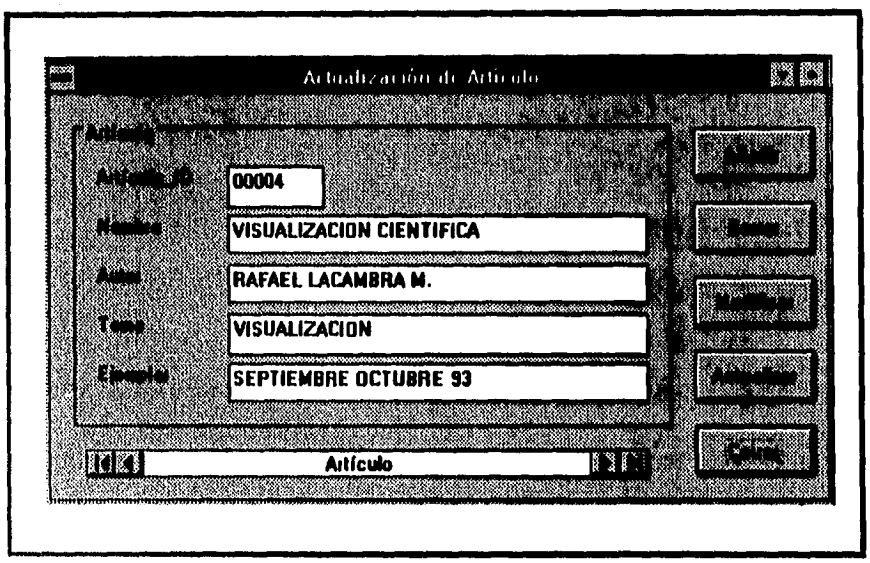


Figura 3.27

Por ejemplo, si seleccionó Artículo, aparecerá la pantalla que se muestra en la figura 3.27, la cual cuenta con comandos de añadir (para dar de alta un artículo), borrar (para eliminar el registro que esté en ese momento en pantalla), modificar (con el fin de corregir algún

registro de la tabla Artículo), actualizar (después de cada operación, para que quede salvado en la base de datos los cambios realizados), y finalmente el comando cerrar, como su nombre lo indica cierra la ventana de actualización. Para accesar nueva información deben ser unicamente mayúsculas para evitar inconsistencias en la base de datos. En caso de error, aparecerá un mensaje y cancelará la última acción.

De manera similar son las actualizaciones de la tabla Revista.

Otra alternativa para llegar a las pantallas de actualización es oprimiendo Control+R para la tabla Revista y Control+A para la tabla Artículo.

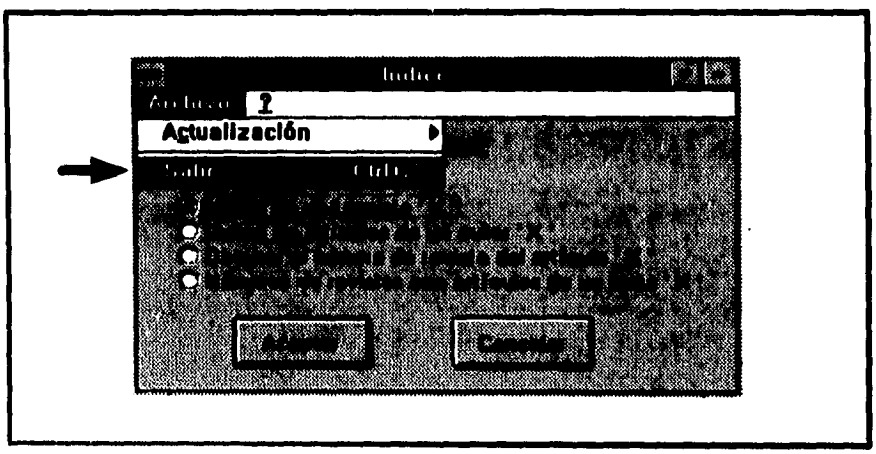


Figura 3.28

Para terminar la aplicación se debe estar en la pantalla principal y realizar cualquiera de las siguientes acciones:

- Hacer doble clic al cuadro superior izquierdo para cerrar la ventana principal
- Seleccionar la acción 'Salir' del menú Archivo (figura 3.28).
- Oprimir Control+Z

Y es así como queda conformada la implementación del sistema Indice, junto con su funcionamiento a grandes rasgos.

Conclusiones

Después de llevar a cabo el presente trabajo se puede concluir lo siguiente:

- Una de las principales ventajas que ofrece el paradigma Orientado a Objetos es su nivel de abstracción, ya que éste es a nivel del mundo real, sin preocuparse en un principio por aspectos que involucren la implementación, eliminando de esta manera limitaciones en el análisis de un problema.

- La visión que se debe tomar al utilizar cualquier paradigma Orientado a Objetos para resolver un problema es viéndolo como un conjunto de objetos que interactúan entre sí.

- Debido al enfoque empleado en sistemas desarrollados bajo este paradigma, se incrementa la flexibilidad de estos sistemas, es decir, facilita los futuros cambios necesarios para darles mantenimiento.

- Otra gran ventaja que ofrece el paradigma OO es la reutilización de código, la cual se manifiesta de tres distintas maneras. La primera se presenta en todas aquellas instancias que pertenecen a una misma clase debido a que comparten igual estructura y comportamiento. La segunda es mediante herencia, ya que todas las subclases heredan los mismos atributos y métodos de su(s) superclase(s). Finalmente la tercera manera de reutilización de código, aunque un poco más difícil, es la creación de clases de objetos lo suficientemente generales para que puedan ser utilizadas en otras aplicaciones.

- **El llevar a cabo un análisis y diseño Orientado a Objetos no implica que forzosamente tenga que estar implementado con herramientas OO, sino que permite utilizar otro tipo de herramientas como estructurales, funcionales o basadas en objetos.**

- **La metodología OMT además de contar con todas las ventajas que ofrece este paradigma, permite diseñar bases de datos relacionales a partir de un análisis OO. Aunque existen varias maneras de mapear objetos, asociaciones y herencias a tablas, es papel del diseñador decidir entre eficiencia y rapidez de consulta. Ya que si se emplea una manera de mapear una asociación, se garantiza no haber violado la tercera forma normal, y si se opta por otra lo que se obtiene es mayor rapidez de consulta. Es por esto que se considera que la estructura de las bases de datos resultantes al utilizar esta metodología es la misma en eficiencia que se obtiene al emplear cualquier otra, pero con la ventaja de que al llevar a cabo el diseño de esta a través de OMT, se cuenta con la facilidad en el análisis debido al nivel de abstracción.**

- Entre las ventajas que ofrece **Visual Basic de Microsoft es la rapidez y facilidad de** crear aplicaciones así como **la disminución de código para el desarrollo de** interfaces.

- En cuanto a las desventajas de **Visual Basic para el desarrollo de sistemas, es que** se necesita una máquina con bastante **memoria, y además, se trata de un ambiente** basado en objetos y no orientado **a ellos, el cual no permite crear nuevos objetos,** sino únicamente los ya existentes.

Bibliografía

[Enstminger-94] Enstminger, Gary.
Secrets of the Visual Basic 3 Masters
Segunda Edición
Sams Publishing 1994.

[Greiff-93] Greiff, Warren R.
El Ocaso del Sol a través de una ventana.
Soluciones Avanzadas
Septiembre-Octubre 1993, año 1, no. 5.

[Hergert-93] Hergert, Douglas A.
Visual Basic 3.0 Programming
Segunda Edición
Random House 1993.

[Ibargüengoitia-92] Ibargüengoitia G., Guadalupe E.
¿Qué significa el Paradigma Orientado a Objetos en Computación?
Sistemas
Agosto 1992, año 1, no. 1.

[Ibargüengoitia-94] Ibargüengoitia G., Guadalupe; López G., Amparo.
Análisis Orientado a Objetos Mediante la Técnica de Modelado de Objetos OMT (Segunda Parte)
Soluciones Avanzadas
Agosto 1994, año 2, no. 14.

[Lopez-94] López G., Amparo; Ibargüengoitia G., Guadalupe.
Análisis Orientado a Objetos Mediante la Técnica de Modelado de Objetos OMT (Primera Parte)
Soluciones Avanzadas
Agosto 1994, año 2, no. 10.

[Morales-92] Morales Gamboa, Rafael.
El Paradigma O2 en el Desarrollo de Sistemas (Primera Parte)
Sistemas
Agosto 1992, año 1, no. 1.

[Oktaba-93a] Oktaba, Hanna.
¿Por qué está de moda la Orientación a Objetos?
Sistemas
Agosto 1993, año 1, no. 2.

[Oktaba-93b] Oktaba, Hanna.
Programación Orientada a Objetos: ¿Moda o Realidad?
Soluciones Avanzadas
Abril-Mayo 1993, año 1, no. 3.

[Oktaba-93c] Oktaba, Hanna; Quintanilla, Gloria.
Abstracción de Datos en Lenguajes Orientados a Objetos
Soluciones Avanzadas
Septiembre-Octubre 1993, año 1, no. 5.

[Orozco-93] Orozco y Orozco, Octavio.
Nuevas Tecnologías para el Desarrollo de Aplicaciones
Soluciones Avanzadas
Noviembre-Diciembre 1993, año 1, no. 6.

[Pressman-88] Pressman, Roger S.
Ingeniería del Software, un enfoque práctico
Segunda Edición
McGraw Hill, 1988.

[Quintanilla-93] Quintanilla, Gloria; Silva, Sergio.
Elementos de la Programación Orientada a Objetos
Soluciones Avanzadas
Julio-Agosto 1993, año 1, no. 4.

[Ramírez-93] Ramírez Ramírez, Miguel.
OO Los Objetos están más cerca que nunca
Soluciones Avanzadas
Noviembre-Diciembre 1993, año 1, no. 6.

[Rumbaugh-91] Rumbaugh, James; Blaha, Michael; Premerlan, William; Eddy, Frederick; Lorensen, William.
Object Oriented Modeling and Design
Prentice Hall, Inc. 1991.

[Sanchez-93] Sánchez Aguilar, Antonio.
Postulados de los Paradigmas de Programación
Soluciones Avanzadas
Septiembre-Octubre 1993, año 1, no. 5.