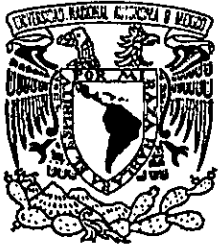


9
2000



UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

OMT CON IMPLEMENTACION
EN DELPHI

257831

T E S I S

QUE PARA OBTENER EL TÍTULO DE
MATEMÁTICO CON ESPECIALIDAD
EN COMPUTACIÓN

P R E S E N T A :

DELIBES FLORES ROMÁN



DIRECTOR DE TESIS:
MA. GUADALUPE E. IBARGÜENGOITIA GONZÁLEZ

1998

REGISTRO DE TESIS
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO



Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.



UNIVERSIDAD NACIONAL
AUTÓNOMA DE
MÉXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo de Tesis:

OMT. con implementación en Delphi

realizado por Delibes Flores Román

con número de cuenta 8955553-4 , pasante de la carrera de Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis
Propietario

M. en C. Ma. Guadalupe Ibargüengoitia González

Ma. Guadalupe Ibargüengoitia

Propietario

Mat. Victor Hugo Dorantes González

Propietario

Mat. Juan Jesús Gutiérrez García

Suplente

Mat. Ma. de los Angeles Bautista Zamudio

Suplente

Ing. Lucila Margarita Urrutia

Juan Jesús Gutiérrez García
Bautista Zamudio
L. Urrutia

Consejo Departamental de Matemáticas

Mat. César de Valencia Savo

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL
DE
MATEMÁTICAS

AGRADECIMIENTOS

- A mi directora de Tesis

M.C. Ma. Guadalupe E. Ibargüengoitia González

Por compartir sus conocimientos, su tiempo, experiencia y amistad

- A mis Sinodales:

Mat. Víctor Hugo Dorantes

Fis. Juan Jesús Gutiérrez

Mat. Ma. De los Angeles Bautista

Ing. Lucila Margarita Cortina

Por sus sugerencias y tiempo dedicado a la revisión de éste trabajo.

DEDICATORIA

- A mi padre

Por ser la primera enseñanza en mi vida, sus valiosos consejos y apoyo.

- A mi madre

Por su amor paciencia y cuidados.

- A mi hermana Enselmini

Por su apoyo y confianza en todo éste tiempo.

Indice general

INTRODUCCION.....	pág. III
I ORIENTACION A OBJETOS.....	pág. 1
I.1 Objeto.....	pág. 1
I.2 Clase.....	pág. 2
I.3 Herencia.....	pág. 2
II CONCEPTOS DE BASES DE DATOS RELACIONALES.....	pág. 4
II.1 Conceptos generales de los sistemas manejadores de bases de datos	pág. 4
II.2 Modelo de tres vistas.....	pág. 5
II.3 Conceptos.....	pág. 6
III DESARROLLO ORIENTADO A OBJETOS MEDIANTE LA TECNICA DE MODELADO A OBJETOS OMT.....	pág. 7
III.1 Técnicas del modelado de objetos.....	pág. 7
III.2 Fase de análisis en OMT.....	pág. 8
III.2.1 Modelado de objetos.....	pág. 8
III.2.2 Modelo dinámico.....	pág. 10
III.2.3 Modelo funcional.....	pág. 13
III.3 Diseño.....	pág. 15
III.3.1 El diseño arquitectónico.....	pág. 15
III.3.2 Diseño de objetos.....	pág. 15
III.4 Bases de datos.....	pág. 16
III.5 Mapeo de clases a tablas.....	pág. 17
III.5.1 Mapeando Asociaciones binarias a tablas.....	pág. 19
III.5.2 Mapeo de generalizaciones a tablas.....	pág. 21
III.6 Implementación.....	pág. 21
IV DELPHI	
IV.1 ¿ Que es Delphi ?.....	pág. 22

V PLANTEAMIENTO DE UN PROBLEMA.....	pág. 26
V.1 Análisis.....	pág. 26
V.2 Diseño.....	pág. 33
V.3 Implementación.....	pág. 40
V.3.1 Interfaz humana.....	pág. 42
V.3.2 Clases persistentes.....	pág. 47
CONCLUSIONES.....	pág. 52
Apéndice A.....	pág. 53
BIBLIOGRAFIA.....	pág. 61

INTRODUCCIÓN

La finalidad de ésta tesis es hacer una aplicación utilizando programación orientada a objetos, para lo cual nos basaremos principalmente en el libro de Rumbaugh OMT con siglas en inglés (Object Modeling Technique) para hacer el análisis, diseño e implementación.

Se mostrará paso a paso el desarrollo de un programa, que en éste caso se trata del control de una red de videocentros.

Para lo cual en el primer capítulo haremos un breve repaso de la metodología orientada a objetos, en el segundo capítulo daremos a conocer la metodología OMT de Rumbaugh y en el tercer capítulo trataremos el problema del videocentro el cual analizaremos, diseñaremos e implementaremos basándonos en los pasos descritos en el capítulo dos.

Nos hubiera gustado realizar el programa utilizando un sistema manejador de bases de datos orientado a objetos, pero nos fue difícil conseguirlo por lo que haremos la implementación en un sistema manejador de bases de datos relacional como propone Rumbaugh en el capítulo 17.

En este caso utilizaremos Delphi por ser un lenguaje orientado a objetos con un ambiente para la programación muy agradable y amigable, y que además cuenta con algunas herramientas para crear bases de datos relacionales (Desktop Database), algunos objetos para hacer los enlaces con la base de datos (Tquery, Ttable, etc.) y para las interfaces para crear menús, ventanas, etc. permitiendo la creación de un sistema interactivo de manera fácil.

El trabajo presentado está estructurado de la siguiente manera:

- En el capítulo I algunos conceptos de orientación a objetos.
- En el capítulo II daremos algunos conceptos sobre bases de datos relacionales.
- En el capítulo III daremos la metodología OMT.
- En el capítulo IV hablaremos un poco sobre Delphi.
- Y en el capítulo V desarrollaremos una aplicación utilizando la metodología OMT.

CAPITULO I

ORIENTACION A OBJETOS

A la orientación a objetos se le define informalmente como una disciplina de modelado y desarrollo de software, que facilita la construcción de sistemas complejos viéndolos como una colección de objetos que se comunican.

Los aspectos fundamentales del paradigma de orientación a objetos son: *características de los objetos, clases, y la herencia.*

I.1 Objeto

¿Que es un *objeto*? Existen muchas definiciones de objeto, como la encontrada en [Booch 91,p77] : "Un objeto tiene estado, comportamiento, e identidad ; la estructura y comportamiento de objetos similares se definen en una clase común; los términos objeto e instancia son intercambiables". Otra definición referida por Booch que incluye a Smith y Tockey: "un objeto representa individual e identificablemente una cosa, unidad o entidad, ya sea real o abstracta, con un rol bien definido en un dominio de problema." Y [Cox 91] : "Cualquier cosa con un contorno bien definido"(en contexto, esto es "fuera del dominio de computación"). [Martin 92, p 204] define: "un "objeto" es cualquier cosa a la que sea aplicable el concepto".[Rumbaugh 91]define: "Definiremos un objeto como un concepto, abstracción o cosa de nuestro entorno y con significado para nuestro problema".

Las características de los objetos, son usadas para describir un conjunto de objetos con la misma representación. Así como las operaciones asociadas para cada tipo abstracto de datos." [Khoshafian.,Object Oriented Databases]

Propiedades de los objetos

- 1.- El objeto es una instancia de una clase; ésta indica el tipo de objeto.
- 2.- El objeto tiene un identificador; un identificador está asociado con el objeto.
- 3.- El objeto tiene un estado; el estado de un objeto es el valor de sus atributos, los atributos el conjunto de tipos de datos admitidos por la clase para sus objetos, y el comportamiento el funcionamiento de los métodos de un objeto.

Identidad de los objetos

Un concepto importante de los objetos es su *identidad*. La identidad es la propiedad de un objeto de distinguirse de entre todos los demás.

El tipo mas común de identidad de un objeto en los lenguajes de programación, bases de datos y sistemas operativos son los nombres definidos para los objetos, a partir de una característica propia del objeto.

- 1.- Un identificador está asociado a un objeto de tipo no base o sea los no dados por el lenguaje de programación como son los enteros, reales, etc.
- 2.- Un identificador es asociado al objeto al momento de su creación y permanece asociado a este sin importarle en que estado esté; ni que modificaciones haya sufrido.

3.- Usando la identidad, los objetos pueden contener o hacer referencia a otros objetos. Esto elimina la necesidad de usar nombres de variables que no involucren a la identidad de los objetos, aunque introduce algunas limitaciones: una de las limitaciones es que un simple objeto puede tener muchas vías de acceso.

1.2 Clase

Una clase es un término que denota clasificación, y también tiene un nuevo significado en los métodos de orientación a objetos. Dentro del contexto OO, una clase es una especificación de estructura (instancias variables), comportamientos (métodos), y herencia (estructura y/o comportamientos de padres) para un conjunto de objetos. Una clase es un descriptor/constructor de objetos.

La clase provee un mecanismo de separación entre la interfaz y la implementación del tipo de datos. Se oculta su representación interna, de los objetos al mundo exterior y protege los algoritmos que implementan comportamientos externos. A esto se le llama encapsulación.

La *clase* es el lenguaje de construcción más comúnmente usado para definir un *tipo abstracto de datos* en lenguajes de programación orientados a objetos.

1.3 Herencia

La *generalización* y la *herencia* son abstracciones para compartir similitudes entre las clases preservando sus diferencias, la generalización es la relación entre una clase y una o más clases de versiones refinadas de ésta; la clase es llamada *superclase* y las refinadas son llamadas *subclases*.

Evitan la necesidad de rediseñar y recodificar, ya que nuevas clases pueden proporcionar sus comportamientos (operaciones, métodos, etc.), así como la representación de atributos de clases existentes. Heredar comportamientos permite compartir código de módulos de software existentes. La representación de una clase padre permite la herencia de estructura de datos de los objetos.

La combinación de estos dos tipos de herencia (representación y comportamientos) permite una estrategia poderosa de modelado y desarrollo de software.

Hay varios temas que son importantes en la tecnología orientada a objetos. Aunque estos temas no son exclusivos de esa tecnología, como son:

Abstracción

La abstracción consiste en centrarse en los aspectos esenciales inherentes de una entidad, e ignorar sus propiedades accidentales. En el desarrollo de sistemas esto significa centrarse en lo que es y lo que hace un objeto antes de decidir como debería ser implementado. El uso adecuado de la abstracción permite utilizar el mismo modelo para el análisis, diseño de alto nivel, estructura del programa, estructura de una base de datos y documentación.

Encapsulamiento.

El encapsulamiento (denominado también el ocultamiento de información) consiste en separar los conceptos externos del objeto, a los cuales pueden acceder otros objetos, de los detalles internos de implementación del mismo, que quedan ocultos para los demás. La encapsulación evita que el programa llegue a ser tan interdependiente que un pequeño cambio tenga efectos secundarios masivos.

Polimorfismo

Polimorfismo... un concepto en tipo teoría, de acuerdo con el cual un nombre (tal como una variable declarada) puede denotar objetos de cualesquiera clases diferentes que estén relacionadas por una superclase común; así de esta manera, cualquier objeto denotado por éste nombre será capaz de responder para un mismo conjunto común de operaciones en diferentes vías. [Booch 91, p. 517]

Significa que una misma operación puede comportarse de modos distintos en distintas clases [Rumbaugh 91, p. 2].

CAPITULO II

CONCEPTOS DE BASES DE DATOS RELACIONALES

En este capítulo describiremos los conceptos más importantes de las bases de datos relacionales.

II.1 Conceptos generales de las Sistemas Manejadores de Bases de Datos (con siglas en inglés DBMS)

Un sistema DBMS, es un programa de computadora para el manejo de un depósito de datos permanente. A éste depósito de datos permanente se le llama **base de datos** y lo constituyen uno o más archivos. Y algunas de las razones de su uso son las siguientes:

- *Recuperabilidad.* La base de datos está protegida contra fallas de hardware, fallas de dispositivos, y algunos otros errores.
- *Compartir entre usuarios.* Múltiples usuarios tienen acceso a la base de datos al mismo tiempo.
- *Compartir aplicaciones.* Múltiples programas, pueden leer y escribir datos en la misma base de datos.
- *Seguridad.* Los datos deben estar protegidos contra los accesos de lectura y escritura.
- *Integridad.* Se pueden especificar reglas que satisfagan los datos. Un DBMS puede controlar la calidad de estos datos, en base a las facilidades provistas por un programa de aplicación.
- *Extensibilidad.* Los datos pueden ser rediseñados en la base de datos sin interrumpir programas existentes. Los datos pueden ser reorganizados para un mejor desempeño.
- *Distribución de datos.* La base de datos puede estar distribuida a lo largo de varios sitios, organizaciones, y plataformas de hardware.

El ciclo de vida de la mayor parte de las aplicaciones de bases de datos incluye los siguientes pasos.

1. Diseño de aplicación.
2. proyectar una arquitectura para acoplarla a la base de datos.
3. Seleccionar una DBMS como servicio de plataforma.
4. Diseñar la base de datos. Escribir el código DBMS para activar las estructuras propias de una base de datos.
5. Escribir programas en un lenguaje de programación para llevar a cabo la interfaz con la base de datos.
6. Llenar la base de datos con información.
7. Correr la aplicación. Consulta y actualizar la base de datos como sea necesario.

La más importante y dificultosa tarea para algunas aplicaciones de bases de datos, es el diseño de la base de datos. El diseño acompañado de código de programación es más fácil. Se debe diseñar una base de datos por las siguientes razones: realizar el diseño antes de codificar mejora la calidad y los costos. Al diseño de bases de datos se le llama *modelo o esquema* de la base de datos.

En general existen dos caminos para diseñar una base de datos. El primero es el manejo de atributos: compilar una lista de atributos importantes para la aplicación y sintetizando grupos de atributos que preservan dependencias funcionales. La otra vía es el manejo de estados: descubrir estados que son significativos para la aplicación y describirlos. En un diseño típico, existen diez veces menos estados que atributos, por lo tanto el diseño de estados es mucho más factible. El modelado de objetos es una forma de diseño de estados.

II.2 Modelo de tres vistas

Es la arquitectura estándar para una familia de aplicaciones de bases de datos relacionales. Esta arquitectura fue originalmente propuesta por el ANSI/SPARC *committee on DBMS*. La idea básica es que el diseño de una base de datos consiste de tres vistas: esquema externo, esquema conceptual, y esquema interno. Cada esquema externo es un diseño de base de datos desde la perspectiva de una simple aplicación. El esquema externo es una visión o abstracción del global.

Las aplicaciones se aíslan en el esquema externo de las modificaciones en el esquema conceptual. El esquema conceptual integra aplicaciones relacionales y oculta las peculiaridades del DBMS. El esquema interno negocia con las limitaciones y características de un específico DBMS. Los niveles de un esquema interno consiste de código del DBMS actual requerido para la implementación del esquema conceptual.

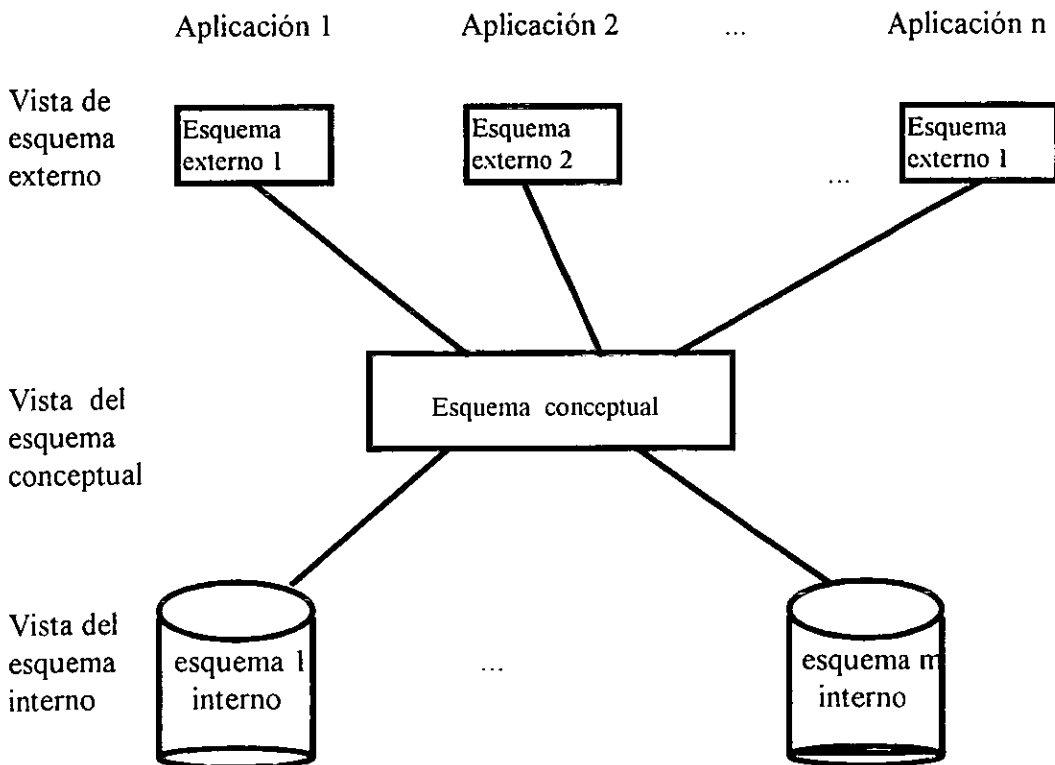


fig. 2.1 Los tres esquemas de arquitectura ANSI/SPARC

II.3 Conceptos de bases de datos relacionales.

El modelo de bases de datos relacionales fue inventado por E.F. Codd y está basado en el concepto matemático de relación. Un manejador de bases de datos relacional (RDBMS) es un programa de computadora para manejar tablas. Y consta de relaciones, conjunto de tablas y llaves. Una relación denota una colección o conexión entre objetos con el mismo tipo de características, los RDBMS son[E.F. Codd]:

- Datos representados como tablas.
- Operadores para manipular las tablas.
- Reglas de integración sobre las tablas.

Una base de datos relacional aparece como una simple colección de tablas. Las tablas tienen un número específico de columnas, y un número arbitrario de renglones. Las columnas de las tablas representan *atributos* y directamente corresponden a los atributos en los modelos orientados a objetos. Los renglones representan instancias y corresponden a las instancias de los objetos. Un valor simple es almacenado en la intersección de cada renglón y columna de la tabla.

La teoría de las bases de datos relacionales dictamina que cada atributo debe estar asignado a un dominio. Un dominio es un conjunto de valores legales. El concepto de dominio es similar al tipo en un lenguaje de programación. Desafortunadamente, la mayoría de los RDBMS no soportan dominios y solo soportan formas simples de datos como números, fecha, y cadenas de caracteres. Cada valor en la tabla debe pertenecer al dominio de estos atributos o debe ser nulo. Ser nulo significa que el valor es desconocido o no aplicable para cierto renglón.

Ahora relacionaremos el modelo de objetos con la arquitectura de tres vistas. Primero formularemos el modelo de objetos para el esquema externo y conceptual, haremos la traducción de cada objeto modelado a tablas ideales. Observaremos los programas de interfaz conectando tablas externas a conceptuales y las tablas conceptuales las convertiremos al esquema interno.

El orden para traducir un objeto a tablas ideales, tiene muchas alternativas de las cuales escoger. Por ejemplo, existen dos vías para mapear una asociación a tablas y cuatro vías para mapear una asociación. Debemos también agregar detalles que no tienen importancia en el modelo de objetos, tales como las llaves primarias y candidatos a llaves para cada tabla y donde un atributo puede ser nulo, los candidatos a llaves usualmente no pueden ser nulos. Debe asignarse un dominio para cada atributo.

El esquema interno consiste de comandos SQL para crear las tablas, atributos, y un buen desempeño.

La estrategia del uso de los IDs (identificadores) en el (RDBMS), es compatible con la notación de *identidad* en los lenguajes orientados a objetos.

CAPITULO III

DESARROLLO ORIENTADO A OBJETOS

MEDIANTE LA TECNICA DE MODELADO A OBJETOS OMT

III.1 Técnica de modelado de objetos OMT

En el paradigma orientado a objetos la idea central es efectuar una abstracción del mundo real en términos de objetos para crear un modelo en donde se identifican y se organizan los conceptos importantes relacionados con la solución del problema a resolver.

Una metodología de desarrollo es un proceso de producción de software que usa una colección de técnicas y una notación especial. Se presenta como una serie de pasos a seguir con la notación asociada a cada uno. Dichos pasos se organizan a lo largo del ciclo de vida del software que comprenden las etapas del análisis , diseño e implementación.

La metodología de modelado de objetos de Rumbaugh, conocida como OMT, se sigue durante el ciclo de vida completo. Tiene un enfoque que difiere de los tradicionales en los siguientes aspectos:

Cambia el esfuerzo de desarrollo al análisis, es decir el análisis es la base más importante en el desarrollo de un sistema. Es el análisis quien capta el mayor esfuerzo, esto tiene ventaja en que el software resulta más claro y adaptable a futuros cambios.

Pone énfasis en la estructura de datos más que en las funciones. Las estructuras de datos y sus relaciones son menos vulnerables a cambios en los requerimientos que las acciones efectuadas en ellos. Se organiza el sistema alrededor de los objetos, lo que proporciona al desarrollo mayor estabilidad. Los otros conceptos como funciones, relaciones y eventos se organizan alrededor de los objetos.

Los pasos importantes seguidos por la metodología OMT durante el ciclo de desarrollo son:

Análisis

El análisis que se inicia con un texto que describe el problema, a partir de tal descripción se construye un modelo del mundo real, que es una abstracción concisa y precisa de lo que debe hacer el sistema. No incluye decisiones que atañen a la implementación. Este modelo captura tres aspectos esenciales del sistema: los objetos y sus relaciones, el flujo de control dinámico y las transformaciones de sus datos.

Diseño

En el *diseño* se obtiene una arquitectura general del sistema. En esta fase se efectúa un cambio en el énfasis ya que además se considera los conceptos computacionales. Los modelo se refinan, organizan y amplían considerando detalles de la implementación.

Implementación

Para la implementación, los modelos obtenidos se pueden traducir a lenguajes con orientación a objetos o sin ella, lenguajes de manejo de bases de datos o hardware. La programación debe ser una tarea simple, casi mecánica pues las decisiones importantes se tomaron en el diseño.

La metodología OMT utiliza tres modelos que combinados proveen de una descripción completa del sistema.

Modelo de objetos

El modelo de objetos, describe la estructura estática de los objetos: su identidad, relaciones, atributos y operaciones. Al construir este modelo el objetivo es abstraer los conceptos de los datos que son más importantes para una aplicación. Se representan gráficamente por medio de diagramas de objetos que definen las clases y los objetos.

Modelo dinámico

El modelo dinámico, captura el aspecto concerniente al tiempo y a la secuencia de operaciones. Se muestra el control sin importar que hacen exactamente las operaciones. Se representa por un diagrama de estados. Las acciones corresponden a las funciones en el modelo funcional. Los eventos son las operaciones en el modelo de objetos.

Modelo funcional

El modelo funcional, describe los aspectos relacionados con las transformaciones de los valores. Captura lo que hace el sistema sin importar cuándo o cómo. Se representa por diagrama de flujo de datos.

III.2 Fase de Análisis en OMT

El análisis empieza con una descripción del problema generada por el cliente y posiblemente con la participación de los desarrolladores. La descripción puede ser incompleta o informal. Como resultado del análisis se hace más precisa y clara, se exponen ambigüedades e inconsistencias que se resuelven conforme se va clarificando el problema.

III.2.1 Modelado de objetos

El siguiente paso en el análisis, consiste en construir el modelo de objetos, tomando la información necesaria de la descripción del sistema, los conocimientos del dominio de la aplicación y del conocimiento del mundo real. Los pasos para construir el modelo de objetos son:

- i) Identificar las clases.
- ii) Preparar un diccionario de datos.
- iii) Identificar asociaciones entre objetos.
- iv) Identificar atributos.
- v) Organizar y simplificar clases usando herencia.
- vi) Verificar que existan trayectorias de acceso para preguntas probables.
- vii) Iterar y refinar el modelo.

i) Identificar clases y objetos

- i.1) Seleccionar todos los sustantivos pues son candidatos a ser clases. Para esto en la descripción del problema se pueden subrayar los sustantivos y se construye una lista de ellos.
- i.2) La notación para las clases es un cuadrado: **Persona** y para los objetos es óvalo: **Juan**
- i.3) Se eliminan las clases innecesarias por ser redundantes, irrelevantes o vagas, o bien por ser atributos, operaciones o construcciones de implementación

ii) Preparar un diccionario de datos

Las palabras aisladas pueden tener diversas interpretaciones. Así que es conveniente preparar un diccionario que defina con mayor precisión cada clase obtenida en el paso anterior.

iii) Identificar asociaciones.

Una asociación es cualquier dependencia entre dos o más clases y en general se encuentran en las frases de la definición del problema. Para identificar asociaciones se debe:

- 1) Extraer todas las frases de la descripción del problema que impliquen relaciones entre clases, las relaciones se expresan por líneas uniendo a las clases.
- 2) Descartar las asociaciones innecesarias e incorrectas
- 3) Hacer más específica la semántica de las asociaciones, para ello es conveniente agregar nombres de roles donde sea apropiado pues estos describen el papel que juega una clase en la asociación. Para ello la metodología provee una notación gráfica.

Por ejemplo, la frase “una empresa contrata a un empleado” se representar con el rol:

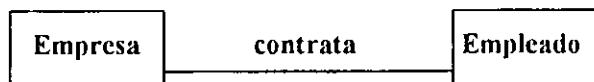


fig. 3.1 una asociación con rol

- 4) Expresar la multiplicidad correcta. Esta puede ser “uno a uno”, representada por una línea recta; “uno a muchos”, representada por una línea recta con una bolita rellena a lado de “muchos”; o “muchos a muchos” representada con bolitas rellenas en cada extremo de la línea.

iv) Identificar Atributos

Los atributos son propiedades de los objetos tales como nombres, peso, velocidad, color. Es poco probable que los atributos aparezcan en la descripción del problema, pero se obtienen del conocimiento del problema en el mundo real.

Para representar los atributos en el diagrama de objetos se debe colocar una línea horizontal separando el nombre de la clase y los atributos. La siguiente figura muestra a la clase “persona” con sus atributos.

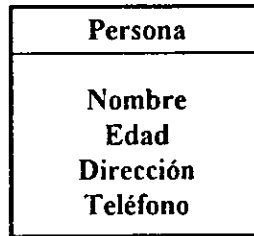


Fig. 3.2 Clase Persona

v) Refinar usando herencia

La herencia puede usarse para generalizar los aspectos comunes en las clases existentes construyendo una superclase o refinarlas en subclasses especializadas.

La representación para la herencia es un triángulo debajo de la superclase. Como se ve en la siguiente figura, una persona puede ser un empleado o un alumno.

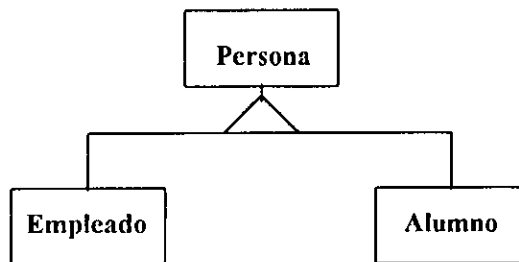


fig. 3.3 Refinamiento de la clase Persona por medio de la herencia

vi) Verificar que existan trayectorias de acceso para preguntas

Este paso consiste en planear preguntas y tratar de contestarlas siguiendo el diagrama de modelo de objetos para ver que se obtengan los resultados esperados.

vii) Iterar y refinar el modelo

Todo es un proceso iterativo, algunos refinamientos pueden darse hasta completar el modelo. Este modelo se refinará en el diseño y luego se implementa.

III.2.2 Modelo Dinámico

En el modelo dinámico se captura el aspecto concerniente a la secuencia de operaciones efectuadas en el tiempo. Se muestra el control sin importar lo que se hace exactamente en las operaciones. El control de un sistema es el aspecto que describe la secuencia de operaciones en respuesta de los estímulos externos y se representa por un diagrama de estados.

El estado de un objeto es el conjunto de valores de sus atributos en un cierto momento, el cual cambia en algún aspecto al recibir un estímulo (dicho estímulo es llamado un evento). En el tiempo van cambiando esos valores pasando el objeto por una secuencia de estados. La respuesta de un objeto a un estímulo depende del estado en que se encuentre.

Un diagrama de estados es una gráfica de estados (representados por círculos) y eventos (representados por flechas) que hacen pasar a los objetos de un estado a otro. El estado inicial se apunta por una flechita y a los estados finales se les pone un círculo negro adentro.

El modelo dinámico consiste de múltiples diagramas de estados uno para cada clase de objetos que tenga comportamiento dinámico importante. Todos los diagramas se combinan en un solo modelo dinámico a través de eventos compartidos entre objetos.

Para construir el diagrama de estados en el modelo dinámico, se comienza con la búsqueda de estímulos y respuestas visibles externamente, los pasos son:

1. Preparar escenarios para las iteraciones típicas.
2. Identificar eventos entre objetos preparando un trazado de eventos para cada escenario.
3. Construir los diagramas de estado.
4. Revisar los eventos entre objetos para verificar su consistencia.

1. Preparar escenarios para una interacción típica.

Un escenario es una secuencia típica de interacciones entre objetos del sistema. Para construirlo se identifican los eventos o acciones típicas de las interacciones de lo que se puede hacer con el sistema. Se deben considerar tanto los casos normales como situaciones en las que ocurran secuencias con acciones de error.

2. Identificar Eventos entre Objetos

Se examinan los escenarios para identificar quien manda y quien recibe las acciones. Se puede mostrar cada escenario como una traza (lista ordenada de eventos entre objetos) Los objetos se muestran en líneas verticales y los eventos como líneas horizontales que unen al objeto que lo envía y lo recibe. Al revisar cada columna de la traza, se puede ver qué eventos afectan directamente a un objeto particular.

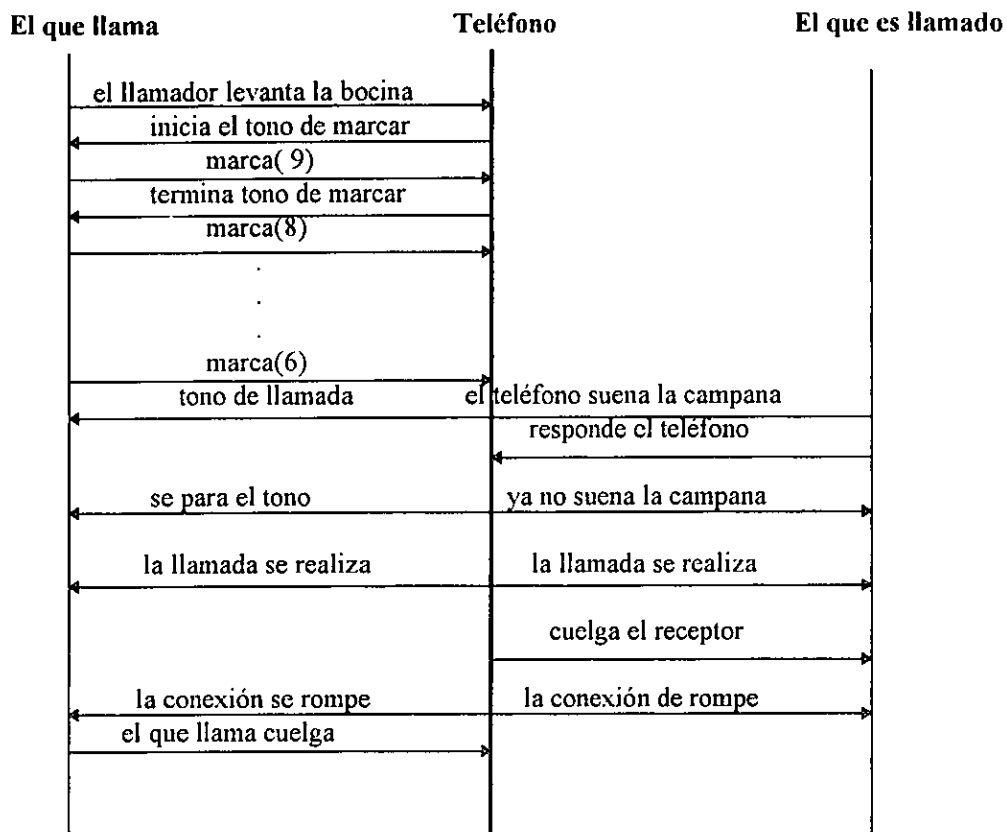


figura 3.4 Traza de eventos para una llamada telefónica.

Se desarrollan trazas para todas las posibles interacciones o casos de uso del sistema.

3. Construir el Diagrama de Estados

Se prepara un diagrama de estados para cada clase de objetos con comportamiento dinámico no trivial, para construirlos se apoya en los objetos que aparecen en las columnas de las trazas, mostrando los eventos que son recibidos y enviados por cada objeto.

Se empieza con una traza que afecte a la clase que se va a modelar. Se toma una interacción típica considerando los eventos que afectan al objeto. Para construir el diagrama de estados, los estados coinciden con las flechas que salen de su columna en la traza y los arcos son las flechas que llegan a su columna y se etiquetan con esa acción.

Después de haber considerado las situaciones normales, se deben agregar los casos especiales y las situaciones de error. El manejo de errores casi siempre complica la estructura de un diagrama limpio y compacto, pero es necesario hacerlo.

El diagrama inicial será una secuencia de eventos y estados, si el escenario puede repetirse indefinidamente, se cierra el diagrama formando un ciclo.

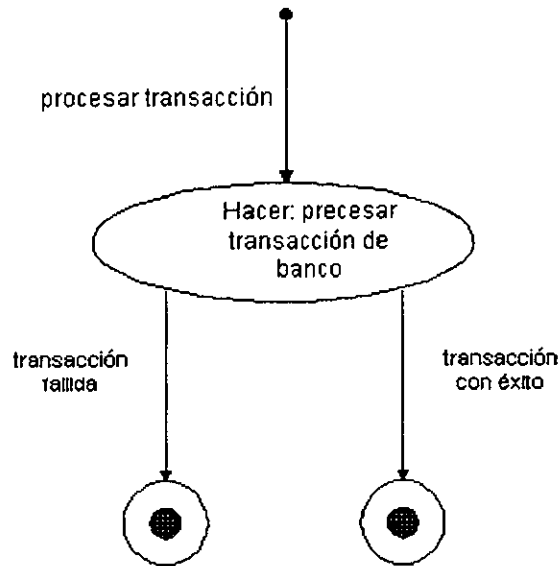


fig. 3.5 Diagrama de estados para una transacción bancaria

4. Revisar los Eventos entre Objetos para verificar su consistencia.

Se deben verificar que todos los diagramas de estados estén completos y sean consistentes entre ellos, esto es, que eventos comunes en distintos diagramas de estados no vayan a estados diferentes.

El conjunto de todos los diagramas de estados para todas las clases de objetos que tienen un comportamiento dinámico importante, constituye el modelo dinámico del sistema.

III.2.3 Modelo Funcional

El modelo funcional especifica el significado de las operaciones en el modelo de objetos y de las acciones en el modelo dinámico. El modelo funcional muestra como se calculan los valores sin importar las decisiones, ni la estructura de los objetos. Utiliza los diagramas de flujo de datos para mostrar las dependencias funcionales.

Un diagrama de flujos de datos es una gráfica que muestra como fluyen los valores de datos a partir de los objetos fuente, a través de los procesos que los transforman hasta sus destinos en otros objetos. Un diagrama de flujo que contiene los procesos que transforman los datos (representados por círculos) flujos de datos (representados por flechas), objetos actores que producen y consumen datos (representados por rectángulos) y objetos que guardan datos pasivamente (representados por líneas horizontales paralelas). En general los diagramas de flujo de datos se construyen por niveles, del más general al más desglosado.

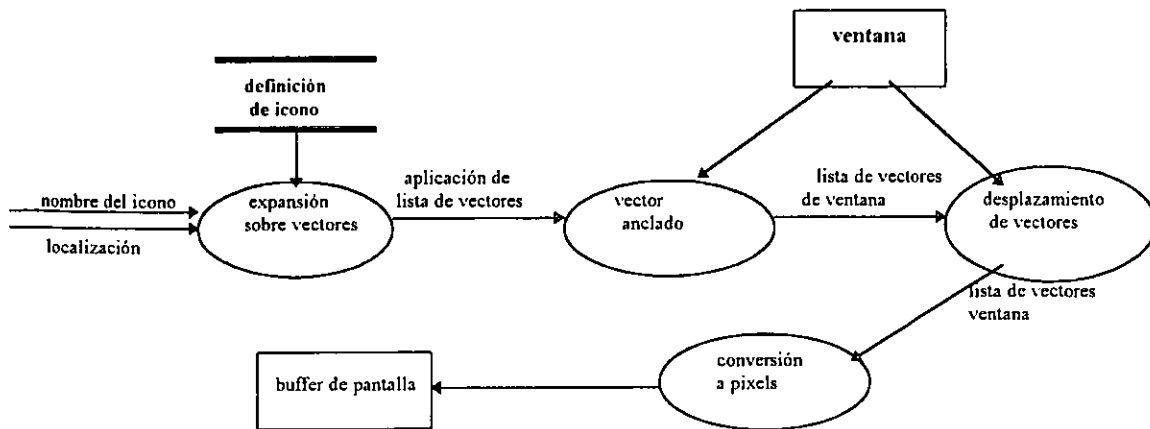


fig. 3.6 Diagrama de flujo de datos para desplegar un gráfico en una ventana.

Muchos sistemas interactivos tiene un modelo dinámico trivial pues su objetivo se reduce a proporcionar y actualizar información, y eso no es dinámico. En contraste en este tipo de sistemas para ese tipo de sistemas el modelo de objetos es importante pues tienen estructuras de datos no triviales.

Los pasos para construir el modelo funcional son:

- 1.- Identificar los actores que producen y consumen datos, las transformaciones y los flujos.
- 2.- Construir los diagramas de flujo de datos mostrando las dependencias funcionales.
- 3.- Describir las funciones.
- 4.- Identificar restricciones.

1.- Identificar los actores

El diagrama de flujo de datos se construye mostrando cómo cada valor de salida se calcula a partir de los valores de entrada. Para cada valor de salida se identifican las secuencias de transformaciones que lo generan desde las entradas.

2.- Construir los diagramas

Construir los diagramas de flujos de datos, que especifican sólo las dependencias entre las operaciones, pero no las decisiones de como efectuarlas en el modelo dinámico donde se muestran las secuencias de las decisiones. En el modelo funcional se desglosan los estados del modelo dinámico.

3.- Describir las funciones

Se describen algorítmicamente cada una de las funciones identificadas en el diagrama de flujos de datos en un lenguaje de descripción, como pseudocódigo, ecuaciones matemáticas, etc. se debe enfocar en qué se hace, no como se implementa.

4. identificar restricciones

Las restricciones son las condiciones de validez específicas para la ejecución del sistema.

III.3. DISEÑO

Presentamos a continuación los pasos para el diseño de sistemas, los cuales no son exclusivos del paradigma orientado a objetos. En el diseño el énfasis cambia para incluir aspectos de la implementación, el diseño se efectúa a dos niveles: Definir la arquitectura y el diseño más detallado al de los objetos en sí.

III.3.1 El diseño arquitectónico consiste de los siguientes pasos:

1. Diseño arquitectónico, la organización de todo el sistema en componentes llamados subsistemas.
2. Se organiza el sistema en subsistemas.
3. Se identifica la concurrencia inherente en el problema.
4. Se asignan los subsistemas a procesadores y a tareas.
5. Se selecciona la estrategia básica para implementar los almacenes de datos en términos de estructuras de datos, archivos y bases de datos.
6. Se identifican los recursos globales y se determinan los mecanismos para controlar el acceso a los mismos.
7. Se selecciona una aproximación para implementar el control del software :
 - Se utiliza la posición dentro del programa para contener el estado, o bien
 - Se implementa directamente una máquina de estados, o bien
 - Se utilizan tareas concurrentes.
8. Se consideran las condiciones de entorno.
9. Se consideran las prioridades de compensación.

III.3.2 Diseño de Objetos

Durante el diseño de objetos, se revisa el modelo de análisis, el cual sirve como base para la implementación. tratando de generalizar la construcción del sistema tomando en cuenta detalles del lenguaje de programación y las bases de datos particulares. El modelo de objetos señala el comienzo del desplazamiento de los objetos del mundo real, al enfoque de la implementación tratando de llevar esto a la fácil implementación en la computadora.

1. Se refina el modelo de objetos buscando en los demás modelos las operaciones que efectúa cada clase.
 - Se buscan las operaciones para cada proceso del modelo funcional y se ponen en la clase apropiada.
 - Se definen las operaciones para cada suceso del modelo dinámico, dependiendo de la implementación del control y se ponen en la clase apropiada.
2. Se diseñan algoritmos para implementar las operaciones de cada clase:
 - Se seleccionan algoritmos que minimicen el costo de implementar las operaciones.
 - Se seleccionan unas estructuras de datos adecuadas para los algoritmos.
 - Se definen nuevas clases y operaciones internas según sea necesario.
 - Se asigna la responsabilidad de aquellas operaciones que no estén asociadas claramente con ninguna clase concreta.
3. Se optiman las vías de acceso a los datos:
 - Se añaden asociaciones redundantes para minimizar los costos de acceso y maximizar la comodidad.
 - Se reorganizan los cálculos para una mayor eficiencia.
 - Se guardan los valores derivados para evitar volver a calcular expresiones complicadas.
4. Se implementa el control del software completando la aproximación seleccionada durante el diseño del sistema.
5. Se ajusta la estructura de clases para incrementar la herencia.
 - Se reorganizan y ajustan las clases y las operaciones para incrementar la herencia.
 - Se abstrae el comportamiento común de los grupos de clases.
 - Se utiliza la delegación para compartir el comportamiento cuando la herencia no sea válida semánticamente.
6. Se diseña la implementación de las asociaciones:
 - Se analiza el recorrido de asociaciones.
 - Se implementa cada asociación como una clase separada, o bien añadiendo atributos cuyos valores sean objetos a una de las clases de la asociación (o a las dos).
7. Se determina la representación exacta de los atributos que son objetos.
8. Se empaquetan las clases y las asociaciones en módulos.

III.4 Bases de datos

El paradigma de orientación a objetos es versátil. No solo provee de bases para diseño de sistemas y código de programación, sino también para diseñar bases de datos. El uso de un diseño orientado a objetos se puede aplicar a cualquier base de datos: Jerárquica, relacional, u orientadas a objetos.

La implementación de los modelos de la orientación a objetos se puede efectuar con bases de datos.

Usaremos los IDs de los objetos como llaves primarias, o sea, cada tabla de una clase derivada tendrá una llave primaria la cual heredará de su tabla padre, así como también todas las tablas que descendan de esta. ésta estrategia es compatible con el lenguaje de notación de orientado a objetos.

III.5 Mapeo de clases a tablas

Cada clase mapeada corresponde a una o más tablas. (También a una tabla pueden corresponder más de una clase.) Los objetos en una tabla pueden estar particionados horizontal y/o verticalmente. Por ejemplo, si una clase tiene muchas instancias las cuales a menudo son referenciadas, partirla horizontalmente no serviría de mucho, porque el acceso seguiría siendo tan lento como el no partirla, debido a que seguiríamos teniendo el mismo número de atributos. Lo que si haría eficiente el acceso sería la partición vertical por que entre menos atributos tiene una tabla mapeada es mas rápido el acceso.

No. Persona	Nombre	Dirección
1	José Martínez	Sn. Matías # 14
2	María Pérez	Av. Tlalpan #1534

Partición horizontal

No. Persona	Nombre	Dirección
999	Carlos López	C. Cuatro # 14

Partición vertical

No. Persona	Nombre
1	José Martínez
2	María Pérez
999	Carlos López

No. Persona	Dirección
1	Sn. Matías #14
2	Av. Tlalpan #1534
999	C. Cuatro # 14

fig. 3.8 Partición horizontal y vertical de una tabla

Para mapear una clase a una tabla, por ejemplo, la clase *Persona* con los atributos: *Nombre* y *dirección*. Se mapea a una tabla en que sus atributos corresponden a las columnas, Y como el modelo relacional necesita de llaves primarias se incorpora un identificador ID como parte de la llave. Como parte de la formulación de la tabla modelo, se agregan otros detalles, por ejemplo especificaremos que ID no puede ser nulo, porque es una llave. Decidiremos si el *Nombre* de persona no debe ser nulo y si es candidato a llave, pero como varias personas pueden coincidir en nombre no lo es. El atributo dirección podría ser nulo. Asignamos el dominio de cada atributo, y se especifica la llave primaria para cada tabla.

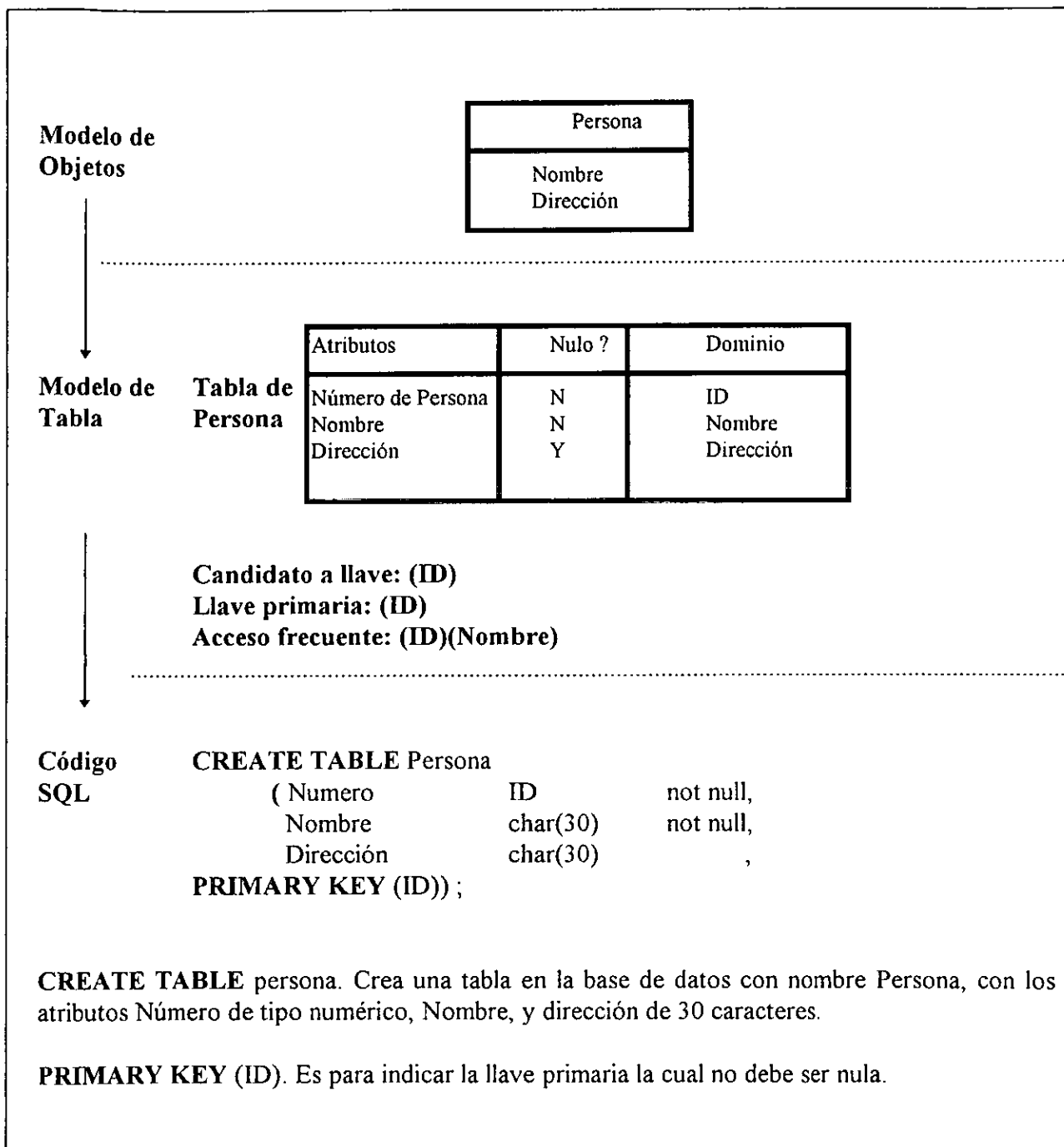


Fig. 3.9 Mapeo de clases a tablas

III.5.1 Mapeando Asociaciones binarias a tablas

En general, una asociación puede o no, mapearse a una tabla. Esto depende de la multiplicidad de la asociación y de las preferencias de los diseñadores de la base de datos acerca de la extensibilidad, número de tablas y el desempeño.

Como la liga entre las tablas, una vez que han sido partidas verticalmente, sirven las llaves primarias las cuales se encuentran en cada tabla que es parte del mapeo de una clase, también para representar la herencia se utiliza este mismo método.

En sistemas grandes con muchas clases es muy pobre el desempeño de este método. La solución a este problema es no particionar en el mapeo de las clases, y las ventajas son:

- Menos tablas.
- Más rapidez de navegación.

Las desventajas:

- *menor apego al diseño*. No se aplica el encapsulamiento. Y en general todos los objetos son accesibles.
- *reduce la extensibilidad*. Esto es debido al diseño.
- *mas complejos*. Una representación asimétrica de la asociación complica la búsqueda y la actualización.

La decisión final sobre si se colapsa una asociación dentro de tablas relacionales depende de la aplicación.

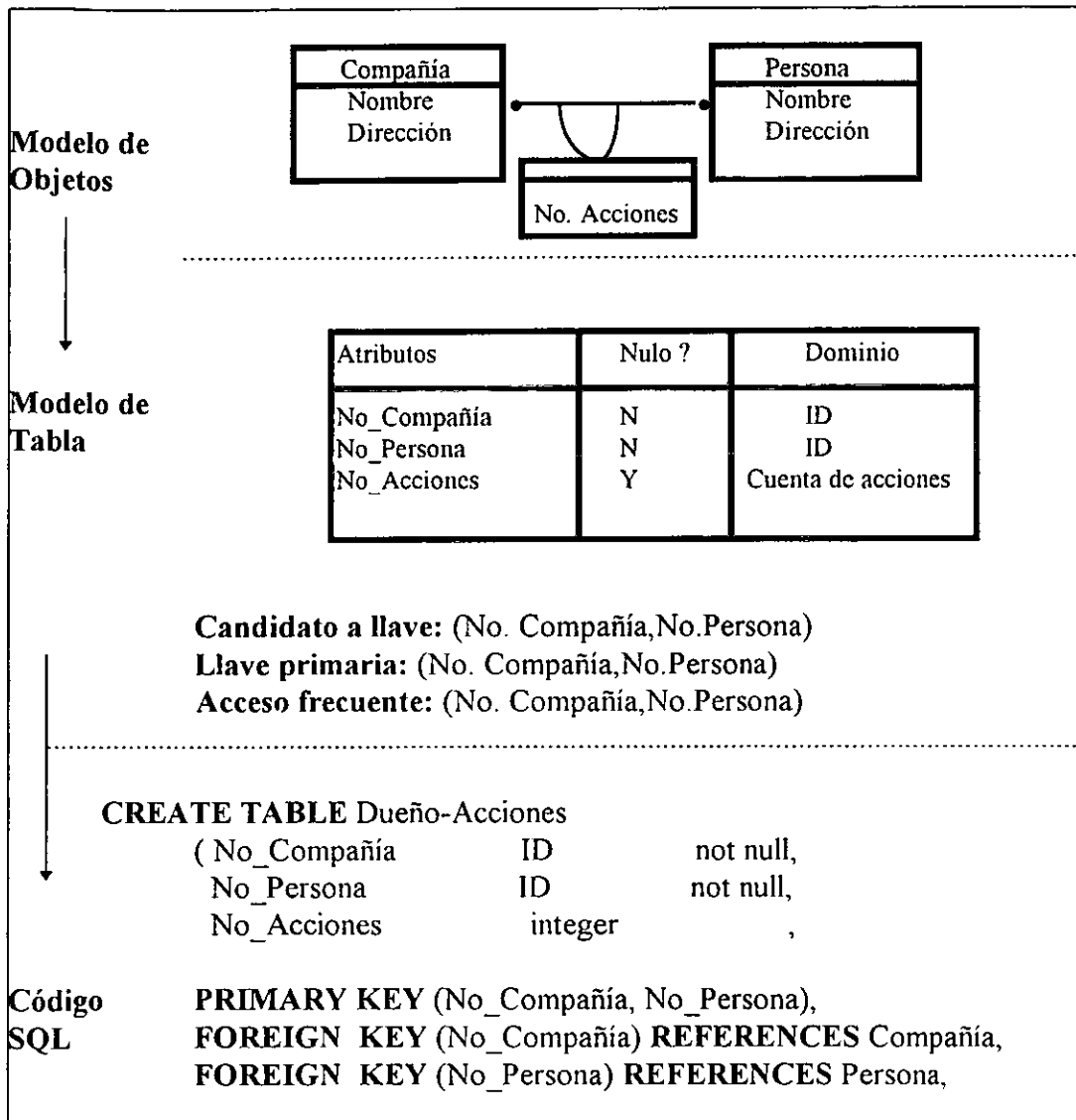


fig.3.10 Mapeo de una asociación muchos a muchos a tablas.

III.5.2 Mapeo de generalizaciones a tablas

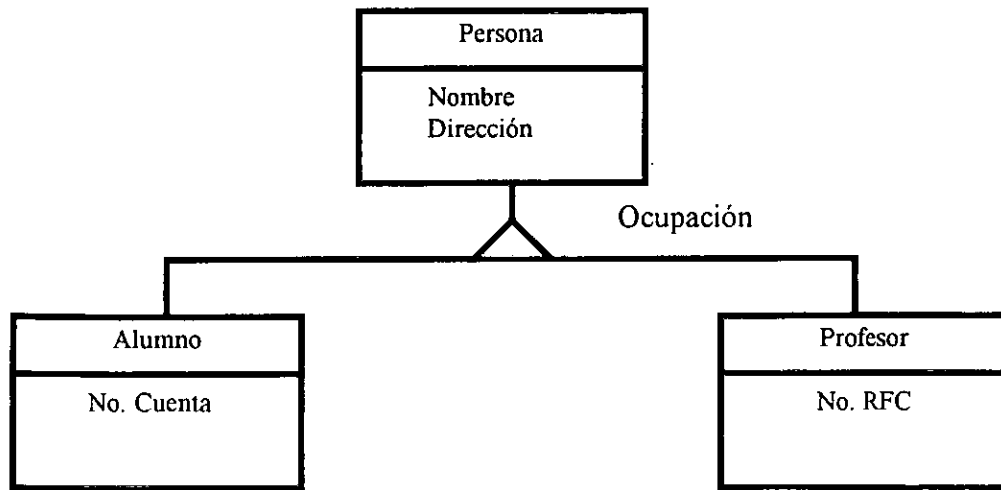


fig. 3.11 Modelo para la generalización

La superclase y las subclases se mapean a tablas. La identidad de un objeto a través de la generalización, se preserva por medio de las ligas que son las llaves primarias, en este caso los IDs, los cuales son los mismos en la superclase y en las subclases correspondientes. Esto es lógicamente limpio y extensible. Si se involucran muchas tablas, y el acceso de la superclase a las subclases es lento. Se pueden embeber las subclases en la superclase con las ventajas y desventajas mencionadas arriba.

III.6 Implementación

La escritura del código es una extensión del proceso de diseño. Por lo que debería ser sencilla, casi mecánica, ya que las decisiones difíciles fueron tomadas durante el periodo de diseño. Y aunque deberemos tomar algunas decisiones durante la escritura del código, esto afecta en pequeña parte al programa. Con todo, el código de programa es la personificación de la solución al problema, así que la forma de escribirlo es importante para la mantenibilidad y extensibilidad.

CAPITULO IV

DELPHI

En éste capítulo daremos una breve descripción de lo que es el lenguaje de programación Delphi, en el cual desarrollaremos nuestro sistema posteriormente.

IV.1 Que es Delphi

Delphi es un ambiente de desarrollo de programas, orientados a objetos. Que contiene Biblioteca de Componentes Visuales (con las siglas en inglés VCL), y posee su propia versión nativa de un (VBX/OCX), denominada componente visual [Visual Component (VC)]. La VCL es el deposito de todos las componentes visuales que puede utilizar el programador de software para crear aplicaciones de Delphi.

Delphi es una poderosa herramienta de desarrollo de programas orientado a objetos que permite la creación de las aplicaciones para windows 3.1/3.11, 95 y NT. Uno de los aspectos más destacados lo constituyen los componentes que Borland ha incluido en Delphi para el desarrollo sencillo y rápido de aplicaciones muy poderosas de bases de datos. No se limita a un formato de datos determinado, si no que se tiene acceso a 50 formatos de datos diferentes a través de controladores suministrados por terceros (IDAPI y ODBC). Entre estos se encuentran los estándares importantes de bases de datos en el área del PC como Xbase, Paradox, Acces, etc. Pero también es posible acceder de forma muy cómoda a servidores de bases de datos de otros sistemas (por ejemplo UNIX) por medio de SQL ("Structured Query Language" cuyo significado es algo así como "lenguaje de consultas estructurado" y que constituye un estándar de lenguaje de uso general, para consultar y modificar datos administrados por servidores especiales de bases de datos como Oracle, Sybase, Informix o Adabas).

Delphi dispone del Object Pascal, un lenguaje de programación compilado muy poderoso que está sin duda a la altura del C++. Este lenguaje surge a partir del desarrollo del Borland Pascal 7.0. El Object Pascal es totalmente compatible con el Borland Pascal 7.0, lo que permite que programas hechos en este lenguaje puedan convertirse a Delphi.

Combina una serie de las mejores propiedades de otros lenguajes de programación y, naturalmente presenta muchas características comunes con ellos. Su ambiente: el entorno de desarrollo completamente integrado, por ejemplo, que no debe abandonarse en ninguna fase de la creación del programa. Editor de código fuente con destacados sintácticos de colores, la visualización de la depuración y la señalización gráfica de los límites de salto. Con un planteamiento visual del entorno de creación.

Las aplicaciones terminadas quedan disponibles como archivos ejecutables (EXE) que pueden utilizarse solos sin bibliotecas adicionales. Consecuentemente, la velocidad con la que pueden ejecutarse los programas creados es muy alta.

Delphi es una Two-Way-Tool, es decir, una herramienta de dos direcciones, porque permite crear herramientas de dos formas: una forma visual en la pantalla, por medio de la función

arrastrar y colocar (Drag & Drop), la otra es a través de la programación convencional, escribiendo el código. Ambas técnicas pueden utilizarse de forma alternativa o simultánea.

Un proyecto Delphi consta de formas, unidades, opciones, configuraciones, recursos, etc.

Los siguientes archivos son creados por Delphi durante el diseño de una aplicación

- **Archivo de proyecto** (project file) (.dpr). Se utiliza para guardar información sobre formas y unidades. También encontrará aquí un código de inicialización.
- **Archivo de Unidad** (Unit file) (.pas). Se utiliza para almacenar código. Algunas unidades se asocian con formas; otras almacenan solo funciones y procedimientos. Muchas de las funciones y procedimientos de Delphi se almacenan en unidades.
- **Archivo de Forma** (Form File) (.dfm). Archivo binario que crea Delphi para almacenar información relativa a sus formas. Cada archivo de forma tiene una unidad (.pas) correspondiente. Por ejemplo, el archivo `miforma.pas` tiene asociados a el archivo denominado `miforma.dfm`.
- **Archivo de Opción de Proyectos** (Project Option Files) (.dfo). En este archivo se almacenan las configuraciones de opciones del proyecto.
- **Archivo de Recursos** (Resource File) (.res). Este archivo binario contiene un icono que usa el proyecto. Este archivo no debe ser modificado o creado por el usuario, ya que Delphi lo crea o actualiza continuamente.
- **Archivos de Respaldo** (Backup files) (~dp, ~df, ~pa). También conocidos como copias de seguridad, éstos son los archivos de respaldo del proyecto, forma y unidades, respectivamente.

El siguiente grupo de archivos son creados por el compilador.

- **Archivo Ejecutable** (Executable File) (.exe). Este es el archivo ejecutable de su aplicación. Es un archivo independiente que no necesita nada más, a menos que utilice bibliotecas contenidas en DLLs, OCXs, etc.
- **Archivo Objeto de Unidad** (Unit Object File) (.dcu). Este archivo es la versión compilada de los archivos de unidad (.pas) y será vinculado con el archivo final ejecutable.
- **Biblioteca de Vinculación Dinámica** (Dynamic-Link-Library) (.dll). Este archivo se crea si usted crea su propia DLL.

Por último, otros archivos windows que pueden utilizarse con Delphi son:

- **Archivos de Ayuda** (Helps files) (.hlp). Estos son archivos windows estándar de ayuda que pueden ser utilizados con su aplicación Delphi.
- **Archivos de Imagen o Gráficos** (Image or graphics files) (.wmf, .bmp, .ico). Estos archivos se usan comúnmente en las aplicaciones windows para construir aplicaciones atractivas y amigables con el usuario.

La Forma

El corazón de toda aplicación de Delphi en Windows es la *forma*. Es posible que se conozca a la forma como una "ventana". La ventana fija es el objeto base sobre el que se construye el resto del ambiente Windows. Cada vez que se inicia Delphi muestra forma en blanco.

Unidad

La unidad es un archivo en lenguaje object Pascal y se utiliza para almacenar código. Algunas unidades se asocian con formas; otras almacenan solo funciones y procedimientos. Muchas de las funciones y procedimientos de Delphi se almacenan en unidades

Recursos y opciones de Delphi

En Delphi encuentran dos tipos de componentes: visuales y no visuales. Los componentes visuales son aquellos que se emplean para construir la interfaz. Ambos tipos de componentes aparecen al momento del diseño, pero los componentes no visuales no son visibles al momento de la ejecución.

La biblioteca (VCL) en Delphi 2.0, se divide en nueve paletas agrupadas de manera lógica : Además hay una ficha OCX y una de Muestras. La ficha Standar, Additional, Win95, Data Acces, Data Control, Win 3.1, Dialogs, System, Qreport, OCX, Samples,

- **La paleta Standar.** Aquí se encuentran los componentes de uso más común. Y es la ficha por omisión al iniciar Delphi.
- **La paleta additonal.** La ficha Additional (adicional) tiene otro grupo de componentes que complementan a los de la ficha estándar.
- **La paleta Win95.** Contiene componentes para crear aplicaciones con el aspecto y la percepción de Windows95.
- **La paleta Data Access.** Contiene componentes para vincularse y comunicarse con la base de datos.
- **La paleta Data Control.** Contiene diversos componentes de atingencia de datos.
- **La paleta Win 3.1.** Almacena componentes que han sido reemplazados por otros más recientes. Se incluyen para fines de compatibilidad con antecedentes de transportar aplicaciones entre Delphi 1.0 y 2.0.
- **La paleta Dialogs.** Contiene componentes que se usan para crear los diversos cuadros de diálogo que son comunes en las aplicaciones Windows.
- **La paleta System.** Contiene componentes para aprovechar las características de Windows.
- **La paleta Qreport.** Contiene componentes para su uso en la generación de reportes.
- **La paleta OCX.** Contiene muestras de controles OCX.
- **La paleta Samples.** Contiene muestras de VCL.

Delphi en sus menú de herramientas ² cuenta con la opción *Database Desktop* con la cual se pueden crear las bases de datos de dos modos, *arrastrar & soltar* o sentencias en SQL. Y para ligarlas con la aplicación Delphi, lo hace a través de la paleta *Data Access*.

Además sus objetos visuales : *query, table y datasource* son muy amigables para hacer el enlace entre Delphi y SQL, Los objetos: *Tedit, DBEdit, ListBox* etc. Son vitales para hacer las consultas. También puede trabajar remotamente las bases de datos (vía red), lo cual es óptimo para hacer una concentración de la información y de manera muy económica. Otra novedad es

que cuando se trabajan localmente las bases de datos se pueden usar alias lo que facilita la programación en cierta forma.

CAPITULO V

PLANTEAMIENTO DE UN PROBLEMA

El objetivo de éste capítulo es: plantear y resolver un problema real utilizando los conocimientos dados en el capítulo III.

Problema: un consorcio de videocentros, necesita un sistema que le ayude al control de prestamos y ventas de películas así como artículos(discos y videojuegos), también para llevar el control de los Socios, Empleados y Clientes de dicho Consorcio.

Se requiere que: ayude a controlar los empleados que tiene cada tienda, el supervisor es quien puede dar de alta o baja un empleado y/o una película. También necesita saber con los socios que cuenta a los que da de alta el supervisor.

A la tienda pueden acudir los socios a rentar o devolver películas y videojuegos. Los clientes son personas que ocasionalmente acuden a comprar algunos de los artículos que se tienen que son: películas, discos o videojuegos.

Un uso básico del sistema es que: llegan los socios del videocentro a la tienda toman la(s) película(s) y solicitan préstamo. El empleado necesita registrarle a cada socio la película(s), así como también mostrarle el precio de cada una de las películas y el total de renta, así como la venta de otros artículos existentes los cuales pueden ser películas especiales para la venta, videojuegos y discos. Cada tienda tiene un dueño o un gerente el cual elige quienes tendrán clave de supervisor.

El sistema debe apoyar en el control de todas estas operaciones.

V.1 Análisis

Siguiendo los pasos de OMT conforme al capítulo anterior (*i*) *identificación de objetos*) se obtuvieron los siguientes candidatos a clases :

1.- Video Centro	X
2.- Empleado	√
3.- sistema	X
4.- películas	√
5.- Socios	√
6.-Artículos	√
7.-Clientes	X
8.-Tienda	√
9.-Supervisor	√
10.-videojuego	√
11.-Discos	√
12.-Consorcio	X

Las clases que tienen (X) son eliminadas por ser vagas o redundantes para la función de nuestro sistema.

Además encontramos las siguientes clases para la interfaz humana las cuales se tratarán al llegar a la etapa de implementación.

- Ventana Principal
- Ventana de Renta
- Ventana de Devolución
- Ventana de Ventas
- Ventana de captura de Socios
- Ventana de captura de Empleados
- Ventana de captura de Artículos
- Ventana de captura de Tiendas

En seguida se identifican los atributos de cada clase y las operaciones que debe tener cada clase según se dijo en el capítulo anterior, posteriormente se identifican asociaciones entre clases lo que nos da el modelo de objetos de la figura 1.

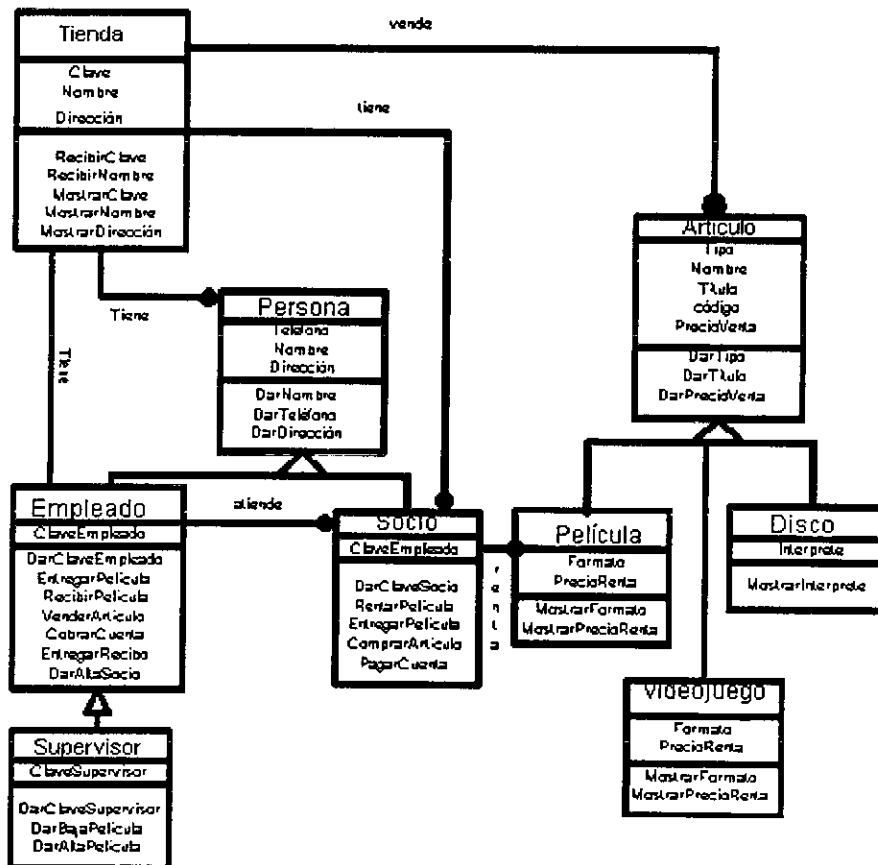


fig.5.1 Modelo de objetos para el sistema.

La clase Tienda se relaciona con la clase Artículo a través de la asociación *vende*, artículo es la clase más general de película, disco y videojuego. La clase tienda esta relacionada con las clases: Empleado, Socio y Supervisor que es una generalización de Empleado, los cuales están generalizados por medio de la clase Persona.

Modelo de interfaz humana también se les identifica los atributos y operaciones. Su diagrama de clases se muestra en la figura 2.

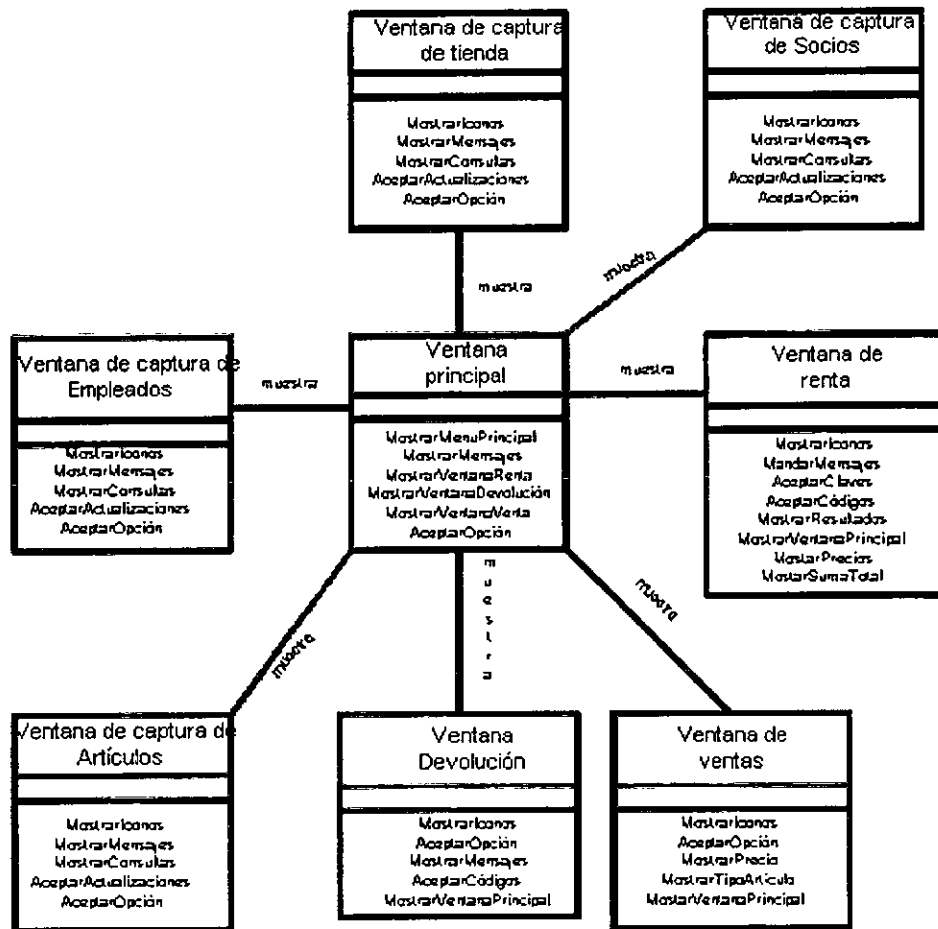


fig.5.2 Modelo de clases para la interfaz humana

La clase Ventana principal como se muestra ésta figura está relacionada a través de *muestra* con todas las demás clases, o sea ella (ventana principal), es la liga entre todas las ventanas.

Para hacer el modelo dinámico se buscan los casos de uso del sistema y se describen por medio de escenarios (aquí solo mostraremos un ejemplo de los casos de cuando existe alguna excepción):

Escenario típico de renta de película.

El socio de la Tienda lleva al Empleado las películas que desea rentar.
El Empleado Elige Rentar en la Ventana Principal.
El Empleado da la clave del Socio a la ventana de renta.
El empleado teclea los códigos de las películas a la ventana de renta.
La pantalla muestra el precio de cada película y el total a pagar.
El Empleado cobra la cuenta al Socio.
El Empleado Elige ventana principal.

Un escenario de renta de películas con excepciones.

El Socio lleva al Empleado las Películas que desea rentar.
El Empleado Elige rentar en la pantalla principal.
El Empleado teclea la clave del socio erróneamente.
La pantalla de renta manda mensaje de clave inexistente.
El Empleado Teclea nuevamente la clave del Socio.

Escenario normal de una devolución de películas.

El Socio Entrega las películas al Empleado.
El Empleado Elige ventana de Devolución.
El Empleado da el código de película a la ventana de devolución.
La ventana de devolución envía mensaje de devolución realizada.
El Empleado Regresa a ventana principal.

Escenario de una devolución de película con excepciones.

El Socio Entrega las películas al Empleado.
El Empleado elige devolución en la ventanaprincipal.
El Empleado Teclea el código erróneamente de la película.
La ventana de Devolución manda mensaje de película inexistente.
El Empleado teclea nuevamente el código.
La ventana de devolución envía mensaje de aceptación.
El Empleado elige ventana principal.

Se hace lo mismo con ventas.

A continuación se pueden mostrar los escenarios de manera mas gráfica por medio de una traza(ver capítulo anterior, como se muestra el las figuras 3, 4 y 5).

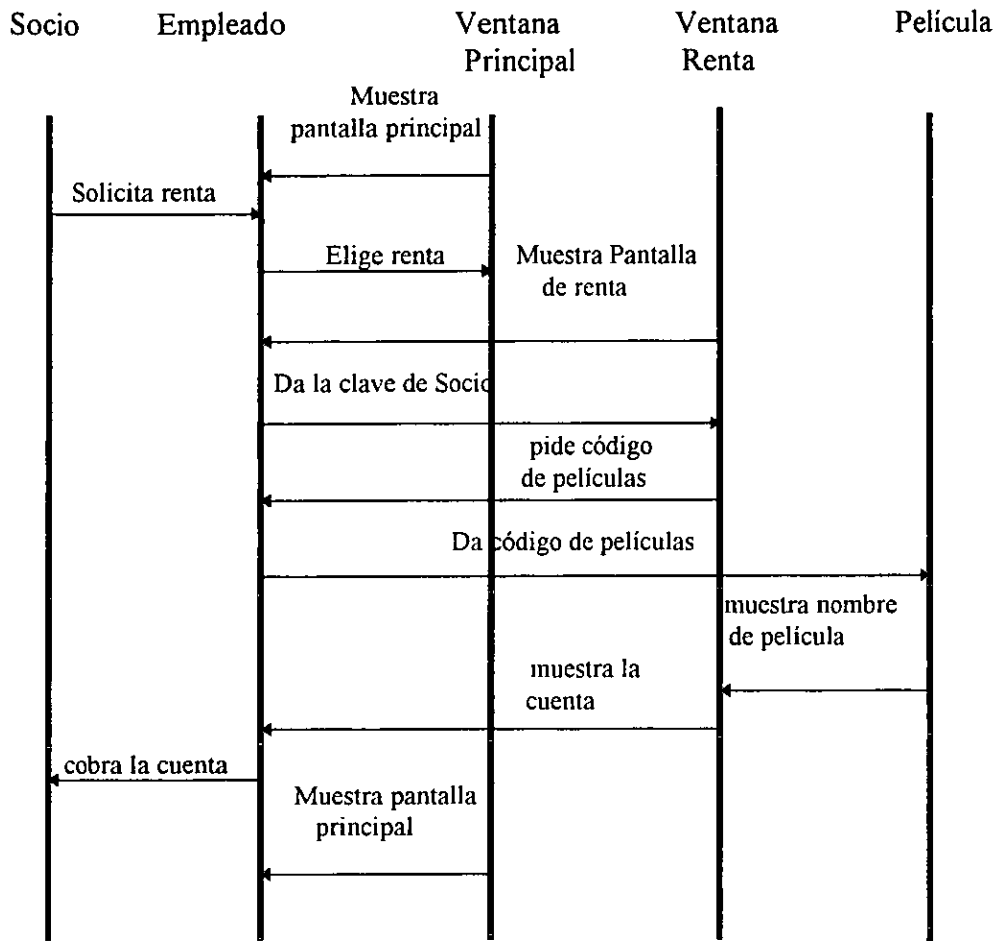


fig.5.3 Traza de sucesos de renta de películas

La ventana principal se muestra. Llega el socio a rentar una película y el empleado elige la opción de renta en la pantalla principal, quien pide a la ventana de renta que se muestre. Esta ventana solicita la llave del socio, el empleado teclea la clave del socio. La ventana de rentas pide los códigos de las películas a rentar, el empleado da los códigos de las películas a la pantalla de renta, ésta muestra el nombre de las películas, su precio y el total de venta. El empleado cobra la cuenta al socio y regresa a la pantalla principal.

De igual manera se hacen las siguientes trazas:

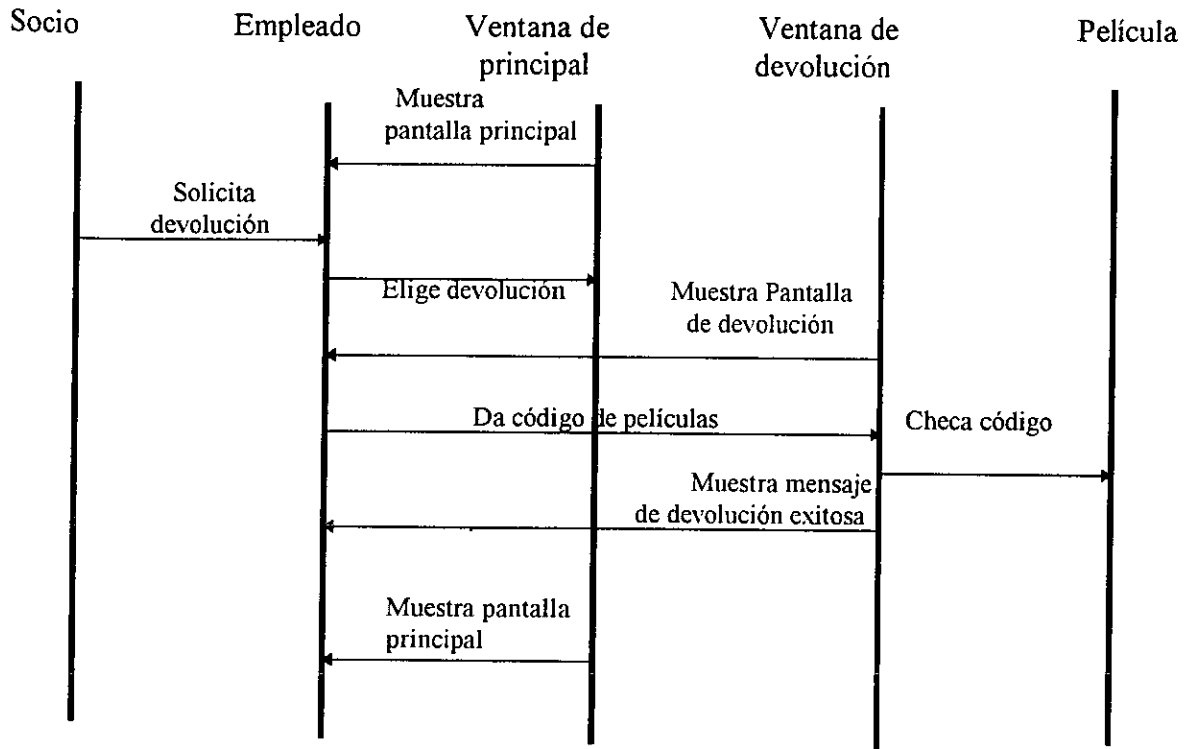


fig.5.4 Traza de sucesos de devolución

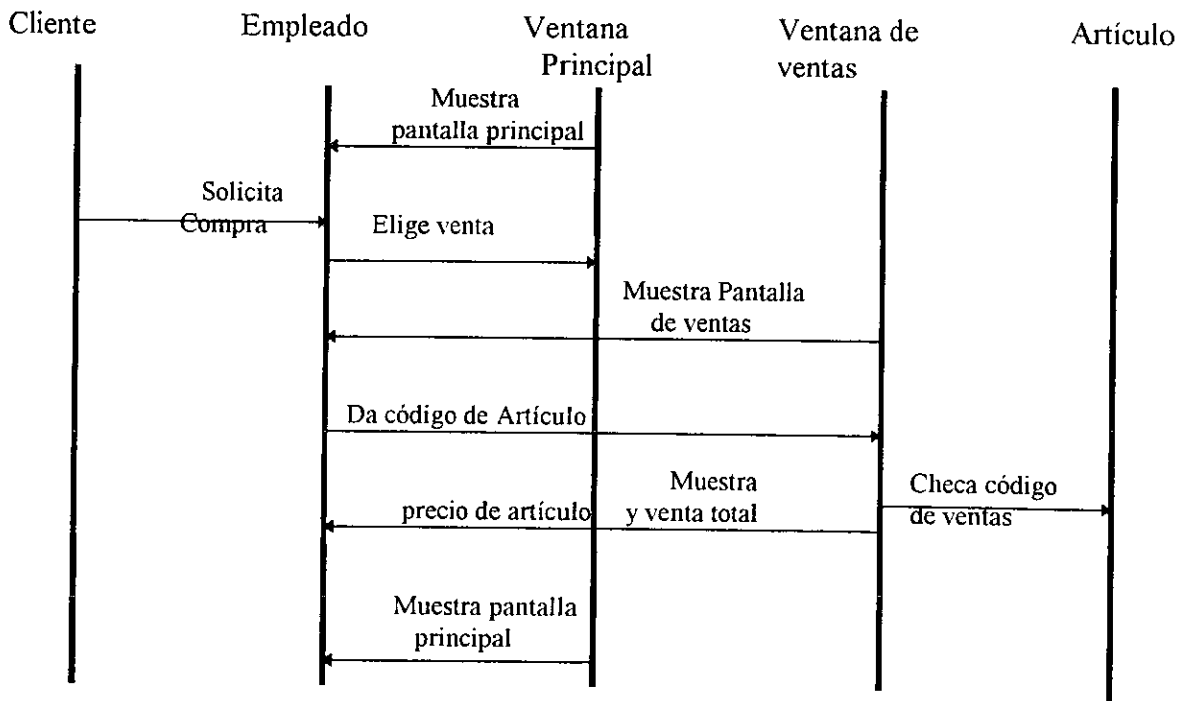


fig. 5.5. Traza de sucesos para la compra de artículo

El paso (3) del modelo dinámico requiere la construcción de un diagrama de estados.

Debido a que todas las clases presentan un comportamiento dinámico o de control trivial, no es necesario hacer el diagrama de estados.

Para *el modelado funcional*.

Diagrama de flujo de datos de nivel para las transacciones se presentan en la figura 6.

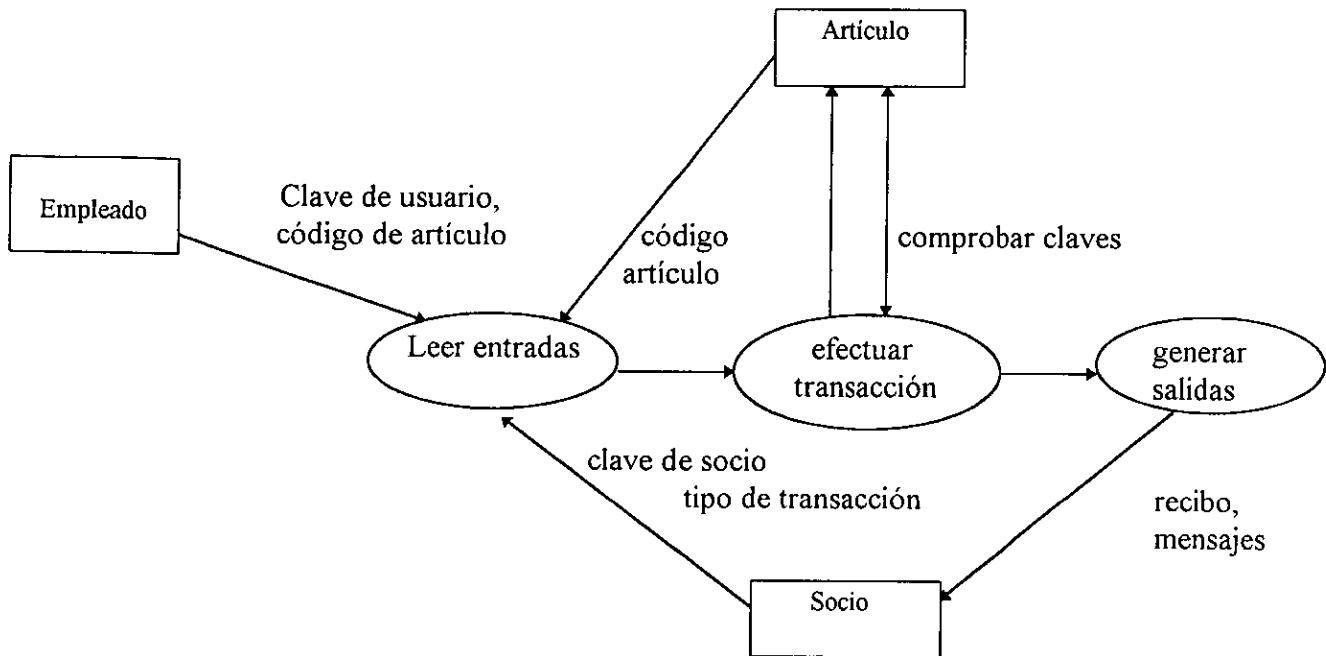


fig. 5.6. Diagrama para el modelo funcional.

La función para leer entradas, efectuar transacción y generar salidas son las funciones que componen nuestro sistema.

3. *Descripción de funciones*, como la mayoría de las funciones del videocentro son triviales, solo describiré la de renta de película, que muestra lo que sucede en todos los demás casos.

Haremos la descripción de la función renta de películas, las funciones devolución y ventas se describen de la misma forma.

RentaPelícula(claveSocio, CódigoPelícula) -> Película, Precio, CuentaTotal, Recibo, Mensaje.
Si la Clave del Socio no existe rechazar la transacción, mandar mensaje .
Si la Clave del Socio existe, teclear Códigos de películas, mostrar precios, y sumar el total de todas las películas rentadas.
El recibo muestra, fecha, hora, clave del Socio, importe y fecha de devolución.

V.2 Diseño

Ver la sección del capítulo II para diseñar una base de datos relacional por lo tanto aplicaremos el capítulo II.5.

El sistema que se está desarrollando es de tipo interactivo y por lo pronto se instalará en una sola tienda dejando aunque ya está contemplado el hacerlo distribuido por medio de la clase *tienda*.

Se mapean a tablas solamente las clases que tienen información que debe ser persistente, o sea las clases: tienda, persona, empleado, socio, artículo, supervisor, disco, película, videojuego.

Aclaración: El mapeo de las siguientes tablas es siguiendo al pie de la letra nuestro modelo de objetos, mas éste no será nuestro diseño final.

MAPEO DE CLASES A TABLAS Modelo de Objeto para Tienda

TIENDA
Clave de la tienda Nombre de la tienda Dirección

Modelo de Tabla para Tienda

Nombre del Atributo	¿Nulo?	Dominio
Clave de la Tienda	No	Ti_clave(ID)
Nombre de la Tienda	No	Ti_nombre
Dirección	No	Ti_direccion

Candidato a llave: (Ti_clave)

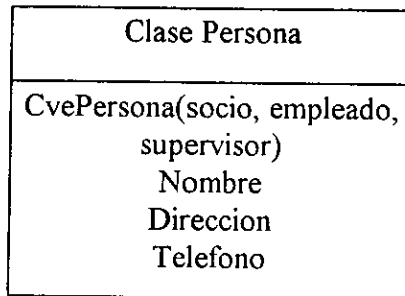
Llave primaria: (Ti_clave)

Acceso Frecuente: (Ti_clave), (ti_nombre)

En cada tabla se ponen como atributos, los atributos de la clase pero se incluye un identificador que sirva como llave en la tabla y también para la herencia se utiliza para ligar las clases padres con las clases hijas.

Aquí cabe señalar que mapeamos las generalizaciones de la clase Persona y Artículo de acuerdo con el modelo de objetos que desarrollamos arriba.

Modelo de Objetos para la clase Persona



Modelo de Tabla para la clase persistente Persona

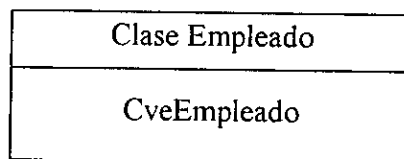
Nombre de Atributo	¿Nulo?	Dominio
ClavePersona (socio, empleado, supv)	No	Pr_Clave(ID)
Nombre de Empleado	No	Pr_Nombre
Direccion	No	Pr_Direccion
Teléfono	Si	Pr_Teléfono

Candidata a llave: (Pr_Clave)

Llave primaria: (Pr_Clave)

Acceso frecuente (Pr_Clave), (Pr_Nombre)

Modelo de Objetos para la clase Empleado



Modelo de tabla para la clase persistente empleado

Nombre de Atributo	¿Nulo?	Dominio
Clave del empleado	No	Em_Clave(ID)

Candidato a llave: (Em_Clave)

Llave Primaria: (Em_Clave)

Acceso Frecuente: (Em_Clave), (Em_Nombre)

Modelo de Objetos para la clase Supervisor

Clase Supervisor
CveSupervisor

Modelo de tabla para la clase persistente empleado

Nombre de Atributo	¿Nulo?	Dominio
Clave del Supervisor	No	Su_Clave(ID)

Candidato a llave: (Su_Clave)

Llave Primaria: (Su_Clave)

Acceso Frecuente: (Su_Clave), (Su_Nombre)

Modelo de Objetos para la clase Socio

Clase Socio
Clave de Socio

Modelo de Tabla para la clase persistente Socio

Nombre de Atributo	¿Nulo?	Dominio
Clave del Socio	No	So_Clave(ID)

Candidato a llave: (So_Clave)

Llave Primaria: (So_Clave)

Acceso Frecuente: (So_Clave), (So_Nombre)

Modelo de Objetos para la clase Artículo

Clase Artículo
<p>Código Tipo Título Nombre PrecioVenta</p>

Modelo de Tabla para la clase persistente Artículo

Nombre de Atributo	¿Nulo?	Dominio
Código	No	Ar_Código
Tipo	No	Ar_Tipo(ID)
Nombre	Si	Ar_Nombre
Título	Si	Ar_Título
Precio de Artículo	No	Ar_Precio

Candidato a llave: (Ar_Código)

Llave Primaria: (Ar_Código)

Acceso Frecuente: (Ar_Código)

Modelo de Objetos para la clase Película

Clase Película
<p>Código PrecioRenta Formato</p>

Modelo de Tabla para la clase persistente Película

Nombre de Atributo	¿Nulo?	Dominio
Código	No	Pe_Código(ID)
PrecioRenta	Si	Pe_Precio
Formato	Si	Pe_Formato

Candidato a llave: (Pe_Codigo)

Llave Primaria: (Pe_Codigo)

Acceso Frecuente: (Pe_Codigo), (Pe_Nombre)

Modelo de Objetos para la clase Disco

Clase Disco
Código Título Precioventa

Modelo de Tabla para la clase persistente Disco

Nombre de Atributo	¿Nulo?	Dominio
Código	No	Di_Código(ID)
PrecioVenta	No	Di_Precio
Interprete	Si	Di_Interprete

Candidato a llave: (Di_Codigo)

Llave Primaria: (Di_Codigo)

Acceso Frecuente: (Di_Codigo), (Di_Título)

Modelo de Objetos para la clase VideoJuegos

Clase VideoJuegos
Código PrecioRenta

Modelo de Tabla para la clase VideoJuego

Nombre de Atributo	¿Nulo?	Dominio
Código	No	Vj_Código
PrecioRenta	No	Vj_Precio

Candidato a llave: (Vj_Codigo)
 Llave Primaria: (Vj_Codigo)
 Acceso Frecuente: (Vj_Codigo), (Vj_Titulo)

Si implementamos la generalización como lo mostramos arriba debido a nuestro manejador de base de datos, el desempeño será muy pobre. Por lo tanto utilizando la flexibilidad del paso III.5.1 del capítulo anterior, que dice que entre menos tablas más rapidez de navegación y que a veces es preferible no particionar, aunque con el riesgo de no seguir nuestro diseño a como lo habíamos planeado que es una de las desventajas mencionadas en dicho paso, además da la flexibilidad al programador de apegarlo a sus herramientas con las que cuente, por lo que el mapeo de las tablas de nuestra aplicación queda de la manera siguiente:

Haremos primero el mapeo de la generalización de las clases Empleado y Supervisor..

Nombre de Atributo	¿Nulo?	Dominio
Clave de Empleado	No	Em_Clave(ID)
Tipo de Empleado	Si	Em_Supervisor

Modelo de la Generalización Persona, Empleado y Supervisor a una tabla...

Aquí embeberemos la clase persona en la clase Empleado en la cual ya con anterioridad habíamos embebido la clase supervisor.

Nombre de Atributo	¿Nulo?	Dominio
Clave de Empleado	No	Em_Clave(ID)
Nombre de Empleado	No	Em_Nombre
Dirección de Empleado	Si	Em_Dirección
Teléfono de Empleado	Si	Em_Teléfono
Tipo de Empleado	Si	Em_Supervisor

Candidato a índice (Em_Clave) (Em_Nombre)
 Índice Primario (Em_Clave)
 Acceso Frecuente (Em_Nombre)(Em_Clave)

Modelo de la Generalización Persona y Socio a una tabla

En la tabla siguiente embebimos las clases persona y socio, porque debido a la función de nuestro sistema nos conviene hacerlo de esta manera.

Nombre de Atributo	¿Nulo?	Dominio
Clave de Socio	No	So_Clave(ID)
Nombre de Socio	No	So_Nombre
Dirección de Socio	Si	So_Dirección
Teléfono de Socio	Si	So_Teléfono

Candidato a índice (Em_Clave) (Em_Nombre)

Índice Primario (Em_Clave)

Acceso Frecuente (Em_Nombre)(Em_Clave)

Modelo de Generalización a una tabla de las clases Artículo, Película, Disco y Videojuego.

Nombre de Atributo	¿Nulo?	Dominio
Código del Artículo	No	Ar_Código
Nombre del Artículo	No	Ar_Nombre
Precio del Artículo	Si	Ar_PrecioVenta
Precio de Renta de Película	Si	Ar_PrecioRenta
Formato del Artículo	No	Ar_Formato
Interprete de Disco	Si	Ar_Interprete
Tipo de Artículo	Si	Ar_Tipo

Candidato a índice (Ar_Código) (Ar_Nombre)

Índice Primario (Ar_Código)

Acceso Frecuente (Ar_Código)(Pe_Clave)

V.3 Implementación

Elegimos Delphi para desarrollar nuestro sistema debido a que es un lenguaje de programación orientado a objetos, que permite crear programas ejecutables y es tan rápido como C o C++. A falta de un sistema manejador de base de datos orientado a objetos, teníamos que mapear los objetos a tablas para lo cual Delphi resultó la mejor opción porque maneja diversos tipos de tablas (paradox, xbase, etc.), y cuenta con una buena versión del lenguaje SQL.

Para implementar las bases de datos se utiliza la "*DataBaseDestop*", que es una herramienta de Delphi para desarrollar bases de datos. La cual cuenta con dos tipos de tablas: Paradox y DbaseX. En nuestro caso elegimos Paradox. Esta herramienta es gráfica así que crear los atributos y las llaves es muy fácil.

Para implementar nuestras clases de interfaz humana: A la primera Ventana que nos muestra Delphi en su inicio la llamaremos Ventana Principal, le pondremos los colores, nombre, tamaño, etc. por medio de la herramienta *Object Inspector* (inspector de objetos) el cual sirve para dar propiedades y funciones a todos los objetos Delphi. Y utilizando la paleta "adittional" pondremos los iconos.

Para la Ventana de Rentas, elegimos en el menú la opción "forms" y nos mostrará otra forma en blanco a la cual al igual que a la pantalla principal la decoraremos por medio de las paletas de herramientas y el *inspector de objetos*.

Y lo mismo para las ventanas de devolución, ventas así como las ventanas de captura de datos.

Nuestra aplicación.

Esta aplicación está desarrollada, para el control de rentas, devoluciones, y ventas de películas de una tienda de video. Así como la venta de otros artículos diversos.

Está desarrollada de cierta forma que es de fácil de usar por el empleado de la tienda. Cuenta con ocho pantallas gráficas haciendo la iteración hombre - máquina muy fácil.

Las pantallas del sistema son:

- Principal
- Rentas
- Devolución
- Ventas
- Artículos
- Socios
- Empleados
- Tiendas

Pantalla Principal. Cuenta con un menú que tiene cuatro opciones: Archivo, Rentas, Devolución, Ventas y Salir. La opción Archivo tiene un submenú, el cual sirve básicamente para la actualización de la base de datos, la opción Rentas muestra la pantalla de Rentas, etc

y la opción Salir Termina con la ejecución del programa. Los cuatro iconos : Rentas, Devolución, Ventas, y Salir realizan la misma tarea que en el menú principal.

Pantalla de Rentas. Una vez dada la clave de Socio, hace una consulta a la base de datos mas específicamente a la tabla de Socios, para comprobar si existe, si éste existe muestra en la pantalla sus datos así como las películas si es que este adeuda. Al teclear los códigos de las películas muestra el precio de renta así como la suma total de la renta.

Pantalla de Devolución. Se teclea el código de la película el cual eliminará de la cuenta del Socio, mandando un mensaje de devolución realizada.

Pantalla de Ventas. Se teclea el código del Artículo, ésta mostrará el tipo de artículo el precio y la venta total.

Las pantallas de Artículos, Socios, Empleados, y Tiendas. Solo son visibles a través del menú principal y sirven para dar de altas, bajas, modificaciones, etc. a las base de datos a las cuales les corresponde una tabla. Cada una de estas pantallas cuentan con iconos con imagen y etiqueta sobre la función que realizan (ver las imágenes mostradas más adelante).

Nuestro sistema contiene una unidad por cada ventana arriba mencionada y en el apéndice A encontraremos un ejemplo del programa para rentas.

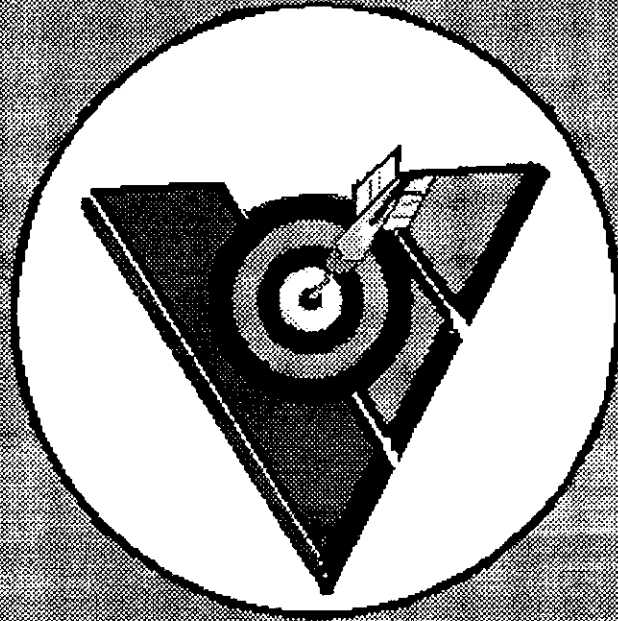
Interfaz humana

- pantalla principal.....pág. 43
- pantalla de rentas.....pág. 44
- pantalla de devolución.....pág. 45
- pantalla de ventas.....pág. 46

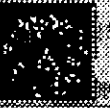

RENTAS


DEVOLUCION


VENTAS



Copyright ...
© 1997
All Rights Reserved..




Consultar...


Limpiar...


Salir...



CLAVE.....:

NOMBRE.....:

DIRECCION...:

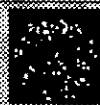
TELEFONO...:

ADEUDOS...:

ARTICULO...:

RENTA.....: \$

CUENTA A PAGAR... \$




SALIR..!


CODIGO..:




VENTA


LIMPIAR


SALIR

Código...

Artículo...

Título.....

Precio.....

Clases persistentes(mapeadas)

- pantalla para captura de artículos
.....Pag. 48
- pantalla para captura de socios
.....Pag. 49
- pantalla para captura de empleados
.....pag. 50
- pantalla para captura de tiendas
.....pag. 51

Consultar... Altas... Bajas... Modificar... Limpiar... Salir...



CODIGO.....: TIPO.....:

FORMATO.....: TITULO.....:

PRECIO DE RENTA.: \$ INTERPRETE.:

PRECIO DE VENTA.: \$ OTROS.....:



Consultar... Altas... Bajas... Modificar... Limpiar... Salir...

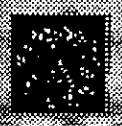



Clave.....:


Nombre....:

Dirección...:

Teléfono....:




Consultar...


Altas...


Bajas...


Modificar...


Limpiar...


Salir...



Clave...







Nombre...

Dirección...

Teléfono...

Supervisor...



 Consultar...	 Altas...	 Bajas...	 Modificar...	 Limpiar...	 Salir...
--	---	---	---	---	---



Clave de Tienda.... :

Nombre:

Dirección:



CONCLUSIONES

El hacer el sistema con la Metodología Orientada a Objetos, OMT implica cambiar la forma de hacer un programa. Se empezó por hacer una entrevista con los empleados del videocentro, Se inicia el análisis con el texto y se construyen el modelo de objetos para la interfaz humana y el dominio del problema, identificando clases siguiendo los consejos dados en la metodología de Rumbaugh (OMT) con la cual ya pude empezar mi análisis. Los escenarios facilitan la comprensión de las funcionalidades del sistema sin pensar en detalles de la implementación que se representan en el modelo funcional. Paso a paso se me facilitaron mucho las cosas, porque a cada día entendía mejor el problema y lo iba modelando.

En el diseño se implementaron las clases del dominio del problema que tenían que ser persistentes y se les mapeo a tablas del modelo relacional.

Llegado el paso del diseño se presentó el problema de no tener un Sistema Manejador de bases de datos Orientado a Objetos, pero como el OMT contempla el mapeo a bases de datos relacionales, se resolvió usando un Sistema Manejador de Bases de Datos Relacional.

Decidimos implementar con Delphi porque reunía todas las características básicas para desarrollar la aplicación interactiva orientada a objetos y con base de datos. Se hacen formas fácilmente con interfaces muy bonitas y cómodas para el usuario final, se crean archivos ejecutables (.exe) que lo hacen portable.

Hacer una consulta, crear una tabla, hacer una actualización me resultó tan fácil e eficiente como hacerlo en Sybase un manejador de bases de datos relacional para sistema operativo Unix el cual había trabajado con anterioridad y con la ventaja de los objetos visuales de Delphi que facilitan aun más la programación de las transacciones.

Los problemas encontrados en Delphi los cuales considero son debidos a que trabajé en la primera versión de este producto son: los datos en la tabla deben tener el tamaño exacto debido a que el sistema (Delphi), no tiene un formato para eliminar los espacios en blanco. Al hacer una consulta y aun poniéndole todas las propiedades al objeto DBEdit, al hacer la consulta no permiten escribir, por lo que es necesario utilizar conjuntamente con estos el objeto Tedit. Y al igual que todos los manejadores de bases de datos que yo conozco, no es claro cuando se compila y manda los mensajes de error del código escrito en SQL, ya que mandan el mensaje (*cerca de y número de línea*), y no en donde está el error lo cual es muy enredoso para los programadores recién iniciados en el lenguaje de bases de datos estándar SQL.

En fin fue muy agradable trabajar con la metodología orientada a objetos, que hace cambiar la forma de plantear un problema, de enfrentarlo y de pensarlo.

Apendice A

Esto es un ejemplo de la implementación del programa *rentas* en Delphi. La parte de arriba es código del esqueleto dado por Delphi. A partir de donde dice implementation están escritos cada uno de mis procedimientos.

```
unit Renta;
```

```
interface
```

```
uses
```

```
SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls, Menus, DB, DBTables, Mask, DBCtrls, Buttons,  
Grids, DBGrids, DBLookup, ColorGrd, ExtCtrls;
```

```
type
```

```
TFRENTAS = class(TForm)  
  Label1: TLabel;  
  CveSocio: TEdit;  
  Label2: TLabel;  
  NomSocio: TEdit;  
  Label3: TLabel;  
  DirSocio: TEdit;  
  Label4: TLabel;  
  TelSocio: TEdit;  
  DBEClave: TDBEdit;  
  DBENombre: TDBEdit;  
  DBEDir: TDBEdit;  
  DBETel: TDBEdit;  
  QueryRentas: TQuery;  
  DataRentaS: TDataSource;  
  Label5: TLabel;  
  Label6: TLabel;  
  CodArticulo: TEdit;  
  ListBox1: TListBox;  
  DBMNomb: TDBMemo;  
  Panel1: TPanel;  
  SpeedButton1: TSpeedButton;  
  SpeedButton5: TSpeedButton;  
  Panel2: TPanel;  
  TipArti: TDBEdit;  
  Label7: TLabel;  
  DBETotal: TDBEdit;  
  Image1: TImage;  
  EditVenta: TEdit;
```

```

Label8: TLabel;
SpeedButton2: TSpeedButton;
procedure MRCONSULTARClick(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure CveSocioKeyPress(Sender: TObject; var Key: Char);
procedure SpeedButton1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
procedure Panell1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
procedure SpeedButton5Click(Sender: TObject);
procedure CodArticuloKeyPress(Sender: TObject; var Key: Char);
procedure SpeedButton5MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
procedure FormHide(Sender: TObject);
procedure SpeedButton2MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
procedure SpeedButton2Click(Sender: TObject);
procedure FormCreate(Sender: TObject);

```

```

private
  { Private declarations }
public
  { Public declarations }
end;

```

```

var
  FRENTAS: TFRENTAS;

```

implementation

```
{ $R *.DFM }
```

```

procedure TFRENTAS.MRCONSULTARClick(Sender: TObject);
Var Sub1,Sub2,sub3,sub4,arreglo,existe,Dato: string;
  i:Integer;
begin
if ((CveSocio.Text <> "") or (NomSocio.text <> "")) then
begin
begin
sub1:='select * from Socios,Articulo ';
sub2:='where (Socios.So_Clave = articulo.Ar_Clave)';
sub3:='and (Socios.So_Clave=:data or Socios.So_Nombre =:Nom)';
arreglo:=sub1+sub2+sub3;
QueryRentas.Active:=False;
QueryRentas.Sql.Clear;

```

```

QueryRentas.Sql.add(arreglo);
QueryRentas.Params[0].AsString:= CveSocio.text;
QueryRentas.params[1].AsString:= NomSocio.Text;
QueryRentas.Active:=True;
end;
if QueryRentas.RecordCount = 0 then
  MessageDlg('No Existe',mtinformation,[MBOK],0)
else
  begin
  QueryRentas.Close;
  QueryRentas.Sql.Clear;
  QueryRentas.Sql.add(arreglo);
  QueryRentas.params[0].AsString:=CveSocio.Text;
  QueryRentas.params[1].AsString:=NomSocio.text;
  QueryRentas.Open;
  DirSocio.Text := DBEDir.Text;
  TelSocio.Text := DBETel.Text;
  QueryRentas.First;
  ListBox1.Items.Add(DBMNomb.Text);
  DbMNomb.Clear;
  For I:=2 to QueryRentas.RecordCount do
  begin
  QueryRentas.Next;
  ListBox1.Items.Add(DBMNomb.Text);
  DbMNomb.Clear;
  end;
  { if CveSocio.Text = " then }
  CveSocio.Text:= DBEClave.Text;
  { else
  CveSocio.Text := QueryRentas.params[0].AsString;}
  { if NomSocio.Text = " then }
  NomSocio.Text := DBENombre.Text;
  { RentaCod.Text:= DBERcod.Text;}
  { else
  NomSocio.Text := QueryRentas.params[1].AsString;}
  end;
  end
else
  MessageDlg('Debes Dar El Nombre o la Clave',mtinformation,[MBOK],0);
end;

procedure TFRENTAS.SpeedButton1Click(Sender: TObject);
begin
FRentas.Hide;
end;

```

```

procedure TFRENTAS.CveSocioKeyPress(Sender: TObject; var Key: Char);
Var Sub1,Sub2,sub3,sub4,arreglo,existe,Dato: string;
    i:Integer;
begin
if key=#13 then
begin
if ((CveSocio.Text <> "") or (NomSocio.text <> "")) then
begin
begin
sub1:= 'select * from Socios left outer join Articulo ';
sub2:= 'on (Socios.So_Clave = Articulo.Ar_Clave)';
sub3:='where (Socios.So_Clave=:data or Socios.So_Nombre =:Nom)';
arreglo:=sub1+sub2+sub3;
QueryRentas.Close;
QueryRentas.Sql.Clear;
QueryRentas.Sql.add(arreglo);
QueryRentas.Params[0].AsString:= CveSocio.text;
QueryRentas.params[1].AsString:= NomSocio.Text;
QueryRentas.Open;
end;
if QueryRentas.RecordCount = 0 then
    MessageDlg('No Existe',mtinformation,[MBOK],0)
else
begin
listbox1.clear;
QueryRentas.Close;
QueryRentas.Sql.Clear;
QueryRentas.Sql.add(arreglo);
QueryRentas.params[0].AsString:=CveSocio.Text;
QueryRentas.params[1].AsString:=NomSocio.text;
QueryRentas.Open;
DirSocio.Text := DBEDir.Text;
TelSocio.Text := DBETel.Text;
QueryRentas.First;
ListBox1.Items.Add(DBMNomb.Text);
DbMNomb.Clear;
For I:=2 to QueryRentas.RecordCount do
begin
QueryRentas.Next;
ListBox1.Items.Add(DBMNomb.Text);
DbMNomb.Clear;
end;
if CveSocio.Text = " then
    CveSocio.Text:= DBEClave.Text
else

```

```

    CveSocio.Text := QueryRentas.params[0].AsString;
    if NomSocio.Text = " then
        NomSocio.Text := DBENombre.Text
    else
        NomSocio.Text := QueryRentas.params[1].AsString;
    end;
end
else
    MessageDlg('Debes Dar El Nombre o la Clave',mtinformation,[MBOK],0);
end;
DBETotal.Text:=' ';
end;
procedure TFRENTAS.SpeedButton1MouseMove(Sender: TObject;
    Shift: TShiftState; X, Y: Integer);
begin
    Panel2.Caption:='Salir de la forma de Rentas ...';
end;

procedure TFRENTAS.Panel1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    Panel2.Caption:="";
end;

procedure TFRENTAS.SpeedButton5Click(Sender: TObject);
Var Sub1,Sub2,sub3,sub4,arreglo,existe,Dato: string;
    i:Integer;
begin
    if ((CveSocio.Text <> "") or (NomSocio.text <> "")) then
        begin
            begin
                sub1:= 'select * from Socios left outer join Articulo ';
                sub2:= 'on (Socios.So_Clave = Articulo.Ar_CLave)';
                sub3:='where (Socios.So_Clave=:data or Socios.So_Nombre =:Nom)';
                arreglo:=sub1+sub2+sub3;
                QueryRentas.Active:=False;
                QueryRentas.Sql.Clear;
                QueryRentas.Sql.add(arreglo);
                QueryRentas.Params[0].AsString:= CveSocio.text;
                QueryRentas.params[1].AsString:= NomSocio.Text;
                QueryRentas.Active:=True;
            end;
            if QueryRentas.RecordCount = 0 then
                MessageDlg('No Existe',mtinformation,[MBOK],0)
            else

```



```

begin
listbox1.clear;
QueryRentas.Close;
QueryRentas.Sql.Clear;
QueryRentas.Sql.add(arreglo);
QueryRentas.params[0].AsString:=CveSocio.Text;
QueryRentas.params[1].AsString:=NomSocio.text;
QueryRentas.Open;
DirSocio.Text := DBEDir.Text;
TelSocio.Text := DBETel.Text;
QueryRentas.First;
ListBox1.Items.Add(DBMNomb.Text);
DbMNomb.Clear;
For I:=2 to QueryRentas.RecordCount do
begin
QueryRentas.Next;
ListBox1.Items.Add(DBMNomb.Text);
DbMNomb.Clear;
end;
if CveSocio.Text = " then
    CveSocio.Text:= DBEClave.Text
else
    CveSocio.Text := QueryRentas.params[0].AsString;
if NomSocio.Text = " then
    NomSocio.Text := DBENombre.Text
else
    NomSocio.Text := QueryRentas.params[1].AsString;
end;
end
else
MessageDlg('Debes Dar El Nombre o la Clave',mtinformation,[MBOK],0);
end;

procedure TFRENTAS.CodArticuloKeyPress(Sender: TObject; var Key: Char);
var
arreglo,sub1,sub2,sub3,sub4,sub5,Consulta: string;
Venta: Real; i,j: integer;
begin
if (key=#13) and (CveSocio.Text<>") then
begin
Venta:=0;
sub1:='update Articulo set Ar_Clave = '"+ CveSocio.Text +'''';
sub2:=' Where Ar_Codigo = :cod';
arreglo:=sub1 + sub2;
QueryRentas.Close;

```

```

QueryRentas.Sql.Clear;
QueryRentas.Sql.Add(arreglo);
QueryRentas.parambyname('cod').AsString:=CodArticulo.Text;
QueryRentas.ExecSql;
if CodArticulo.Text <> " Then
begin
sub3:= 'select Socios.* ,Articulo.* from Socios,Articulo ';
sub4:= 'where Socios.So_Clave = Articulo.Ar_Clave and So_Clave=:Cve';
Consulta:=sub3+sub4;
listbox1.clear;
QueryRentas.Close;
QueryRentas.Sql.Clear;
QueryRentas.Sql.add(Consulta);
QueryRentas.ParamByName('Cve').AsString:=CveSocio.Text;
QueryRentas.Open;
EditVenta.Text:= FloatToStr(Venta);
QueryRentas.First;
ListBox1.Items.Add(DBMNomb.Text);
DbMNomb.Clear;
Venta:=StrToFloat(DBETotal.Text);
For I:=2 to QueryRentas.RecordCount do
begin
QueryRentas.Next;
ListBox1.Items.Add(DBMNomb.Text);
DbMNomb.Clear;
Venta:= Venta + StrToFloat(DBETotal.Text);
end;
end;
EditVenta.Text:= FloatToStr(Venta);
end;
procedure TFRENTAS.SpeedButton5MouseMove(Sender: TObject;
Shift: TShiftState; X, Y: Integer);
begin
Panel2.Caption:='Consulta las rentas realizadas con anterioridad ...';
end;
procedure TFRENTAS.FormHide(Sender: TObject);
begin
CveSocio.Text:="";
NomSocio.Text:="";
DirSocio.Text:="";
TelSocio.Text:="";
CodArticulo.Text:="";
ListBox1.Clear;
DBMNomb.Clear;

```

```
    DBETotal.Text:="";  
    EditVenta.Text:="";  
end;
```

```
procedure TFRENTAS.SpeedButton2MouseMove(Sender: TObject;  
  Shift: TShiftState; X, Y: Integer);  
begin  
Panel2.Caption:='Limpiar la ventana de Rentas ..!';  
end;
```

```
procedure TFRENTAS.SpeedButton2Click(Sender: TObject);  
begin  
  CveSocio.Text:="";  
  NomSocio.Text:="";  
  DirSocio.Text:="";  
  TelSocio.Text:="";  
  CodArticulo.Text:="";  
  ListBox1.Clear;  
  DBMNomb.Clear;  
  DBETotal.Text:="";  
  EditVenta.Text:="";  
end;
```

```
end.
```

BIBLIOGRAFIA

“OBJECT-ORIENTED MODELING AND DESIGN”
JAMES RUMBAUGH, MICHEL BLAHA, WILLIAN PREMERLA.
PRENTICE HALL 1991

“INTRODUCTION TO OBJECT-ORIENTED DATABASES”
KIM, W.
THE MIT Press. 1990

“OBJECT ORIENTED DESIGN WITH APPLICATIONS “
BOOCH, G.
THE BENJAMIN/CUMMNINGS PUBLISHING COMPANY, INC. 1991

“OBJECT-ORIENTED DATABASES”
SETRAG KHOSHAFIAN
JOHN WILEY & SONS, INC. 1993

“DELPHI 2”
DAN OSIER, STEVE GROBMAN Y BATSON
PRENTICE HALL 1996

“SYBASE SQL”
ELISABETH SCOTT
SYBASE INC. 1992