UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

"DESARROLLO E IMPLEMENTACIÓN DEL SISTEMA WEB PARA UN VIDEOCLUB"

T R A B A J O E S C R I T O
EN LA MODALIDAD DE SEMINARIOS Y CURSOS DE ACTUALIZACIÓN Y CAPACITACIÓN PROFESIONAL
QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A:

M I G U E L I N A Y E O P É R E Z

ASESOR: M. en C. JESÚS HERNÁNDEZ CABRERA

MÉXICO, 2007.

DEDICATORIA

"A todos aquellos hombres y mujeres que han hecho del corazón su motorcito interior, porque nunca pierden la fe en Dios y en ellos mismos."

CONTENIDO

INTRODUCCIÓN

I. FUNDAMENTOS PARA EL DESARROLLO EN WEB

II. ANÁLISIS Y DISEÑO CON UML

III. INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

INTRODUCCIÓN

Internet, es el resultado de la interconexión de miles de computadoras en todo el mundo. Todas ellas comparten un mismo lenguaje o protocolo de comunicación.
La World Wide Web (www), mejor conocida como la web, no es más que uno de los muchos servicios que ofrece Internet (de los cuáles profundizaremos en el capítulo I).
Implica ofrecer una interfase simple y consistente para acceder a la inmensidad de recursos de la gran red mundial. Es la forma más moderna de ofrecer información y el medio más potente.

En décadas anteriores, la manera de obtener los datos en Internet era caótica; había un sinfín de maneras posibles y con ello había que conocer múltiples programas, sistemas operativos, etc. Con la web se introdujo un concepto fundamental: la posibilidad de lectura universal, es decir una vez que la información está disponible, se pueda acceder a ella desde cualquier computadora, desde cualquier país, por cualquier persona autorizada, usando un único y simple programa.

Para que esto sea posible, se utilizan una diversidad de conceptos, el más conocido es el hipertexto (documentos que contienen enlaces o links a otros documentos).

El programa que se usa para leer los documentos de hipertexto se llama “explorador” o “browser”. Así, no hay más que buscar la información deseada y comenzar a “navegar” por las diferentes posibilidades que ofrece el sistema. Los exploradores más conocidos son el Explorer de Microsoft, Mosaic y el Netscape Communications. Tienen capacidades diferentes, ésto es muy importante para crear un sistema web, no sólo considerando un buen diseño sino la compatibilidad entre exploradores.

Para desarrollar sistemas en el ambiente de internet se requiere utilizar un conjunto de tecnologías, como por ejemplo: Sistemas Operativos, Lenguajes de Programación, Manejadores de Bases de Datos, etc. Por lo tanto es necesario comprender la misma cantidad de conceptos para poder crear aplicaciones de este tipo.

Sin embargo, cada vez más, surgen aplicaciones y sistemas destinados a funcionar en la web, que resuelven problemáticas específicas, y que han dado excelentes resultados en cuanto a costo de desarrollo, recursos requeridos, así como facilidad de acceso. Por esa razón, son muchas las empresas que se han percatado de estos beneficios y han volteado la mirada hacia éstas aplicaciones para implementarlas en sus ámbitos específicos.

En el presente trabajo de investigación y desarrollo se pretende aplicar todos los conceptos implicados para darle estructura y funcionalidad a un sistema web basado en las actividades propias de un videoclub, con lo cual se busca establecer las pautas a seguir para implementar desde cero un sistema completo en este ambiente.
Para una mejor comprensión del marco teórico, durante el despliegue de la información aquí contenida se irán intercalando aspectos del análisis, desarrollo e implementación del sistema en sí.

En el capítulo I se da un breve pero sistemático bosquejo de las tecnologías involucradas en el desarrollo web.
Posteriormente, en el apartado II se inicia el análisis y diseño del sistema propiamente dicho, utilizando las diversas metodologías y diagramas que ofrece el Lenguaje de Modelado Unificado (UML), así como un abordaje completo de las herramientas de análisis y diseño que ofrece.
Cabe mencionar que la columna vertebral de nuestra aplicación es el lenguaje de programación orientado a objetos Java, por todas las características ventajosas que ofrece, las cuáles se detallan en el capítulo III.

Enseguida, el Modelo de Tres Capas, ocupa nuestra atención en el capítulo IV, desglosando cada uno de sus componentes. Dado que este modelo es el más apropiado para ser usado en una aplicación web.

Y para terminar, en el último capítulo, se presenta un compendio de la implementación del sistema propiamente, integrando los componentes encontrados en el diseño, es aquí donde se unen de manera congruente las clases java necesarias para el control de todos los procesos del sistema. Además, en dicho apartado se encuentran las principales pantallas que estructuran la interfase final con la que interactúa el usuario.

Es importante aclarar, que no es el propósito de este tratado el desarrollo de una aplicación web, en su totalidad, considerando hasta los más mínimos detalles (esencialmente por falta de espacio) sino más bien brindar una guía y descripción de los pasos que deben de seguirse para la creación de un sistema web desde cero, tomando como base las actividades desarrolladas en un videoclub.

I. FUNDAMENTOS PARA EL DESARROLLO EN WEB

INTERNET

Definición

Algunos definen Internet como "La Red de Redes", y otros como "La Autopista de la Información".

Efectivamente, Internet es una Red de Redes porque está hecha a base de unir muchas redes locales de computadoras y porque es la más grande. Prácticamente todos los países del mundo tienen acceso a Internet.

Se dice "navegar" porque es normal el ver información que proviene de muchas partes distintas del mundo en una sola sesión[1].

Una de las ventajas de Internet es que posibilita la conexión con todo tipo de computadoras, desde las personales, hasta las más grandes que ocupan habitaciones enteras. Incluso podemos ver conectados a la Red cámaras de vídeo, robots, etc.

Historia

Internet nació en Estados Unidos hace unos 30 años. Un proyecto militar llamado ARPANET pretendía poner en contacto una importante cantidad de computadoras de las instalaciones del ejército de estadounidense.

Al paso del tiempo, a esta red se fueron añadiendo otras empresas. Así se logró que creciera por todo el territorio de norteamericano. Hace unos 10 años se conectaron las instituciones públicas como las Universidades y también algunas personas desde sus casas. Fue entonces cuando se empezó a extender Internet por los demás países del mundo, abriendo un canal de comunicaciones.

Internet crece a un ritmo vertiginoso. Constantemente se mejoran los canales de comunicación con el fin de aumentar la rapidez de envío y recepción de datos.

Cada día que pasa se publican en la Red miles de documentos nuevos, y se conectan por primera vez miles de personas. Con relativa frecuencia aparecen nuevas posibilidades de uso de Internet, y constantemente se están inventando nuevos términos para poder entenderse en este nuevo mundo que no para de crecer.

[1] Sesión: Un proceso con una serie de peticiones, de páginas visitadas, en el cual un cliente entra a un sitio web, se dirige a la sección A, pide buscar información acerca del producto x. Eso constituye un proceso de navegación, a dicho proceso se le llama sesión.

Estructura

Modelo OSI

El modelo para la Interconexión de Sistemas Abiertos, OSI se ha convertido en una referencia obligada para todo lo relacionado con la intercomunicación de computadoras.

Desde el punto de vista de ISO (Organización Internacional de Normalización), un sistema abierto es el conjunto de una o más computadoras con su software, periféricos y terminales, capaces de procesar y transmitir información.

EL modelo OSI consta de 7 capas:

- Nivel siete: Aplicación

Este nivel se preocupa de proporcionar un conjunto de servicios distribuidos a los procesos de aplicación de los usuarios. El usuario se comunicará directamente con este nivel a través de la correspondiente interfase o agente de usuario. Algunos de esos servicios son servicio de mensajería, servicio de almacenamiento y recuperación de documentos, etc.

- Nivel seis: Presentación

Este nivel se ocupa de la representación de los datos usados por los procesos de aplicación del nivel siete. Por lo tanto, si es necesario, realizará la transformación de los datos que reciba de o para el nivel de aplicación.

- Nivel cinco: Sesión

Su función es establecer y gestionar un camino de comunicación entre dos procesos del nivel de aplicación. Este nivel establece una sesión y se encarga de controlar la comunicación y sincronizar el diálogo.

- Nivel cuatro: Transporte

Este nivel es responsable de una transferencia de datos transparente entre dos entidades del nivel de sesión, liberando a dichas entidades de todo lo referente a la forma de llevar a cabo dicho transporte.

- Nivel tres: Red

Es el responsable del encaminamiento de los paquetes de datos a través de la red. Cada vez que un paquete llega a un nodo[2], el nivel tres de ese nodo deberá seleccionar el mejor enlace de datos por el que envíe la información.

- Nivel dos: Enlace

Un enlace de datos se establece siempre entre dos puntos físicos de conexión del sistema. El nivel de enlace es responsable de la transferencia fiable de cada paquete a nivel de red.

- Nivel uno: Físico

Este nivel engloba los medios mecánicos, eléctricos, funcionales y de procedimiento para acceder al medio físico. Es el encargado de la activación y desactivación física de la conexión.

[2] Nodo: Cada una de las computadoras conectadas a una red.

Protocolo TCP/IP

Un *Protocolo* es un conjunto de reglas que gobiernan las acciones de comunicación.

TCP/IP (Transmission Control Protocol / Internet Protocol) es una familia de protocolos para interconectar computadoras de diversas naturalezas.

Desde su planeación, TCP/IP se pensó para ser independiente del medio físico de enlace, es esto precisamente lo que ha hecho que sea un protocolo ampliamente usado en enlaces de redes locales entre si, o bien, con redes amplias.

Los ambientes que usan TCP/IP se basan en que cada elemento de la red tenga su dirección IP[3]. El propósito de lo anterior es identificar de forma única a cada elemento del conjunto.

TCP tiene 2 funciones importantes:

1. Secuenciamiento y reconocimiento de paquetes.
2. Control del flujo de la información.

Para poder realizar la conexión entre clientes y servidores no importando si éstos están en la misma red o en redes distantes, la solución más sencilla es que ambos: clientes y servidores, se comuniquen usando TCP/IP.

IP y DNS

El protocolo IP hace el trabajo de llevar y traer paquetes entre todas las redes que estén unidas y usando este protocolo, pero no nos garantiza que éstos lleguen a su destino. Para remediar esto, está TCP que nos regula el flujo de paquetes.

Las direcciones de IP tienen como objetivo:

1. Identificar de manera única cada nodo de una red o un grupo de redes.
2. Identificar también a miembros de la misma red.
3. Direccionar información entre un nodo y otro, aún cuando ambos estén en distintas redes.
4. Direccionar información a todos los miembros de una red o grupo de redes.

El *Sistema de Nombre del Dominio (Domain Name System)* se implementó con el fin de brindar un mejor método para relacionar los nombres y direcciones en una red. Los nombres y direcciones se guardan en servidores de nombres (*name servers)*.

[3] Una dirección IP es un número que identifica de manera lógica y jerárquica a una interfaz de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP (Internet Protocol), que corresponde al nivel de red o nivel 3 del modelo de referencia OSI.

Servicios de Internet

Las posibilidades que ofrece Internet se denominan servicios. Cada servicio es una manera de sacarle provecho a la Red independientemente de las demás.
Hoy en día, los servicios más usados en Internet son: Correo Electrónico, World Wide Web, FTP, Grupos de Noticias, Chats.

- La *World Wide Web, o www* como se suele abreviar, se inventó a finales de los 80's en el CERN, el Laboratorio de Física de Partículas más importante del mundo. Se trata de un sistema de distribución de información tipo revista. En la Red quedan almacenadas lo que se llaman Páginas Web, que no son más que páginas de texto con gráficos o fotos. Aquellos que se conecten a Internet pueden pedir acceder a dichas páginas y acto seguido éstas aparecen en la pantalla de su computadora. Este sistema de visualización de la información revolucionó el desarrollo de Internet. A partir de la invención de la www, muchas personas empezaron a conectarse a la Red desde sus domicilios, como entretenimiento. Internet recibió un gran impulso.

- El *FTP*, Protocolo de Transferencia de Archivos (File Transfer Protocol) nos permite enviar archivos de datos por Internet. Con este servicio, muchas empresas informáticas han podido enviar sus productos a personas de todo el mundo sin necesidad de gastar dinero en miles de disquetes ni envíos. Muchos particulares hacen uso de este servicio para dar a conocer sus creaciones informáticas a nivel mundial.

- El *correo electrónico* es una utilidad que permite enviar (o recibir) mensajes a cualquiera de los usuarios de la Red en el mundo. Las ventajas del correo electrónico sobre el correo convencional o las llamadas telefónicas son enormes, la inmediatez es una de ellas. El bajo costo es otro de sus atributos.
  Una última cualidad del correo electrónico es la de ser un mecanismo asíncrono.
  El correo electrónico necesita una dirección electrónica de origen y otra de destino, que puede ser algo así como micuenta@mimail.com. En este caso "micuenta" refiere al nombre del usuario, "mimail" es el nombre del sistema o dominio en el que tiene su cuenta, y "com" quiere decir que el proveedor es una empresa comercial.

- Otro de los servicios es el *Chat*. Cada uno de los participantes ve en la pantalla de su computadora lo que él mismo escribe y lo que responden los demás; en una sesión pueden intervenir varias personas. Para establecer una conversación por este sistema, se debe acordar en que "canal" o chat room se realizará. Ese canal realmente es un espacio de una computadora de Internet, llamada servidor.

- *Trabajo compartido en servidores (SSH). SSH (Secure Shell)* es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos. También permite copiar datos de forma segura. SSH usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión.

Dominios[4] genéricos

.aero	Industria del transporte aéreo
.com	Fines comerciales
.coop	Cooperativas
.info	Información
.museum	Museos
.name	Nombres de personas
.jobs	Departamentos de empleo y recursos humanos en empresas
.net	Infraestructura de red
.org	Organizaciones
.pro	Profesionales
.gov	Gobiernos y Entidades Públicas (En inglés)
.gob	Gobiernos y Entidades Públicas (En español)
.vg	Videogames (En inglés)
.edu	Educación
.mil	Organizaciones militares (Ejército, Armada, Fuerza Aérea)
.int	Internacional, para organizaciones como la ONU
.xxx	Páginas de la industria pornográfica (en tramite de discusión para su aprobación durante el 2007)
.travel	Páginas de la industria de viajes
.mobi	Dedicado a empresas de la rama de la telefonía móvil

Ejemplos de Dominios por distribución geográfica

.es	España
.mx	México
.us	Estados Unidos
.pr	Puerto Rico
.jp	Japón
.fr	Francia
.ca	Canadá

[4] Dominio: Un dominio de Internet es el nombre de equipo que proporciona nombres más fácilmente recordados en lugar de la IP numérica. Permiten a cualquier servicio moverse a otro lugar diferente en la topología de Internet, que tendrá una dirección IP diferente.

TECNOLOGÍAS INVOLUCRADAS EN EL DESARROLLO DE APLICACIONES WEB

¿Qué es HTML?

El HTML (Hyper Text Markup Language) es un lenguaje para estructurar documentos de marcas. Estos documentos pueden ser mostrados por los exploradores de páginas Web en Internet, como Netscape, Mosaic o Microsoft Explorer.

El documento se hallará situado en alguna computadora a la que se pueda acceder a través de Internet.
Para indicar la situación del documento en Internet se utiliza la URL (Uniform Resource Locator).
La URL es el camino que ha de seguir nuestro explorador a través de Internet para acceder a un determinado recurso, bien sea una página Web, un archivo, un grupo de noticias, etc. La estructura de una URL para una página Web suele ser por ejemplo:
http://myp.red.cnc.mx/cine/pelicula.html
Donde:

http://	es el Protocolo
myp.red.cdc.mx	es el Dominio (nombre) de la computadora
/cine/	es el Directorio dentro de la computadora
pelicula.html	es el archivo que contiene la página Web

Servidores WEB

Un servidor Web es una computadora que esta permanentemente conectada a Internet y alberga páginas (comúnmente html), accesibles a todo el mundo.
Un servidor Web se encarga de mantenerse a la espera de *peticiones HTTP*[5] llevada a cabo por un cliente *HTTP* que solemos conocer como explorador. El explorador realiza una petición al servidor y éste le responde con el contenido que el cliente solicita.

Debemos distinguir entre:

- Aplicaciones de cliente: El cliente Web es el encargado de ejecutarlas en la máquina del usuario. Son las aplicaciones tipo Java o Javascript: el servidor proporciona el código de las aplicaciones al cliente y éste lo ejecuta.
- Aplicaciones de servidor: El servidor Web ejecuta la aplicación; ésta, una vez ejecutada, genera cierto código HTML; el servidor toma este código recién creado y lo envía al cliente por medio del protocolo HTTP. Esta estructura suele ser mejor opción a la hora de crear una aplicación Web.

Servidor Apache

Apache es un sistema muy utilizado (actualmente es el servidor más utilizado en Internet). Normalmente se utiliza bajo un sistema operativo Unix o Linux, pero existe una recompilación para win32, aunque no se le considera tan robusto como el apache de Unix.

[5] El protocolo de transferencia de hipertexto (*HTTP, HyperText Transfer Protocol*) es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido.

El servidor Apache es el servicio que se encarga de resolver las peticiones de páginas de Internet de los clientes utilizando el protocolo de Internet http.
Apache está estructurado en módulos. La configuración de cada módulo se hace mediante la configuración de las directivas que están contenidas dentro del módulo. Los módulos de Apache se pueden clasificar en tres categorías:

- Módulos Base: Módulo con las funciones básicas de Apache
- Módulos Multiproceso: Son los responsables de la unión con los puertos de la máquina, acepando las peticiones y enviando a los "hijos" a atender las peticiones.
- Módulos Adicionales: Cualquier otro módulo que le añada una funcionalidad al servidor.

Las funcionalidades más elementales se encuentran en el módulo base, siendo necesario un módulo multiproceso para manejar las peticiones. Se han diseñado varios módulos multiprocesos para cada uno de los sistemas operativos sobre los que se ejecuta Apache, optimizando el rendimiento y rapidez del código.

Servidor de Aplicaciones Tomcat

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP)(Ambos se detallan más adelante) de la empresa desarrolladora de software Sun Microsystems. Tomcat es considerado un servidor de aplicaciones.

El motor de servlets del Tomcat a menudo se presenta en combinación con el servidor web Apache.

Tomcat puede funcionar como servidor web por sí mismo. Opera de tal manera en entornos de desarrollo poco exigentes en términos de velocidad y de manejo de transacciones.

Dado que Tomcat fue escrito en el lenguaje de programación Java, funciona en cualquier sistema operativo que disponga de la máquina virtual.

Tomcat lo desarrollan y lo mantienen miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la Apache Software Licence.

La jerarquía de directorios de instalación de Tomcat incluye:

- bin - arranque, cierre, y otros scripts y ejecutables
- common - clases comunes que pueden utilizar las aplicaciones Web
- conf - ficheros XML y los correspondientes DTDs para la configuración de Tomcat
- logs - logs de las aplicaciones
- server - clases exclusivas del servidor.
- shared - clases compartidas por todas las aplicaciones Web
- webapps - directorio que contiene las aplicaciones Web

Páginas estáticas contra dinámicas

En la Web podemos encontrar, o construir, dos tipos de páginas:

- Las que se presentan sin movimiento y sin funcionalidades más allá de los enlaces.
- Las páginas en las que podemos interactuar y que son generadas en el momento de la petición.

Las primeras páginas son las que denominamos páginas estáticas, se construyen con el lenguaje HTML, que no permite grandes posibilidades para crear efectos ni funcionalidades más allá de los enlaces.

Estas páginas son muy sencillas de crear, aunque ofrecen pocas ventajas tanto a los desarrolladores como a los visitantes, ya que sólo se pueden presentar textos planos acompañados de imágenes y a lo sumo contenidos multimedia como pueden ser videos o sonidos.

El segundo tipo de páginas se denomina página dinámica. Una página es dinámica cuando se incluye cualquier funcionalidad y para ello es necesario utilizar otros lenguajes de programación, aparte del simple HTML.

Para programar una página dinámica necesitaremos otros lenguajes como: C, Java, PHP, ASP, aparte del HTML. Sin embargo, nunca hay que olvidarse del HTML, ya que éste es la base del desarrollo Web: generalmente al escribir una página dinámica el código de los otros lenguajes de programación se incluye embebido dentro del mismo código HTML.

Supongamos que hemos decidido realizar un portal de televisión donde una de las informaciones principales a proveer podría ser la programación semanal. Efectivamente, esta información suele ser dada por las televisoras con meses de antelación y podría ser muy fácilmente almacenada en una base de datos. Si trabajásemos con páginas HTML, tendríamos que construir una página independiente para cada semana en la cual introduciríamos "a mano" cada uno de los programas de cada una de las cadenas televisivas.

Asimismo, cada semana nos tendríamos que acordar de quitar la página de la semana pasada y poner la actual. Todo esto podría ser fácilmente resuelto mediante páginas dinámicas. En este caso, lo que haríamos sería crear un programa (solo uno) que se encargaría de recoger de la base de datos la programación de aquellos programas que son transmitidos en las fechas que nos interesan y de confeccionar una página donde aparecerían ordenados por cadena televisiva y por hora de transmisión. De este modo, podemos automatizar un proceso y desentendernos de un aspecto de la página por unos meses.

Páginas dinámicas de cliente

Son las páginas dinámicas que se procesan en el cliente. En estas páginas toda la carga de procesamiento de los efectos y funcionalidades la soporta el explorador. Usos típicos de las páginas de cliente son efectos especiales para Webs como control de ventanas, presentaciones en las que se pueden mover objetos por la página, control de formularios, cálculos, etc. El código necesario para crear los efectos y funcionalidades se incluye

dentro del mismo archivo HTML y es llamado SCRIPT. Cuando una página HTML contiene scripts de cliente, el explorador se encarga de interpretarlos y ejecutarlos para realizar los efectos y funcionalidades.

Las páginas dinámicas de cliente se escriben principalmente en dos lenguajes de programación: Javascript y Visual Basic Script (VBScript).

Las páginas del cliente son muy dependientes del sistema donde se están ejecutando y esa es su principal desventaja, ya que cada explorador tiene sus propias características, incluso cada versión, y lo que puede funcionar en un explorador puede no funcionar en otro.

Como ventaja se puede decir que estas páginas liberan al servidor de algunos trabajos, ofrecen respuestas inmediatas a las acciones del usuario y permiten la utilización de algunos recursos de la máquina local.

Páginas dinámicas de servidor

Podemos hablar también de páginas dinámicas del servidor, que son reconocidas, interpretadas y ejecutadas por el propio servidor.

Dichas páginas son útiles en muchas ocasiones. Con ellas se puede hacer todo tipo de aplicaciones Web. Desde agendas a foros, sistemas de documentación, estadísticas, juegos, chats, etc. Son especialmente útiles en trabajos donde se tiene que acceder a información centralizada, situada en una base de datos en el servidor, y cuando por razones de seguridad los cálculos no se pueden realizar en la computadora del usuario.

Es importante destacar que las páginas dinámicas de servidor son necesarias porque para hacer la mayoría de las aplicaciones Web se debe tener acceso a muchos recursos externos a la computadora del cliente, principalmente bases de datos. Un caso claro es un banco: no tiene ningún sentido que el cliente tenga acceso a toda la base de datos, sólo a la información que le concierne.

El código de las páginas dinámicas del servidor se suelen escribir en el mismo archivo HTML, mezclado con el código HTML, al igual que ocurría en las páginas del cliente. Cuando una página es solicitada por parte de un cliente, el servidor ejecuta los scripts y se genera una página resultado, que solamente contiene código HTML. Este resultado final es el que se envía al cliente y puede ser interpretado sin lugar a errores ni incompatibilidades, puesto que sólo contiene HTML.

Para escribir páginas dinámicas de servidor existen varios lenguajes. Common Gateway Interface (CGI) comúnmente escritos en Perl, Active Server Pages (ASP), Hipertext Preprocesor (PHP), y Java Server Pages (JSP).

Las ventajas de este tipo de programación son que el cliente no puede ver los scripts, ya que se ejecutan y transforman en HTML antes de enviarlos. Además son independientes del explorador del usuario, ya que el código que reciben es HTML fácilmente interpretable.

Introducción a las bases de datos

Base de datos.- Colección de datos relacionados y estructurados.
Una base de datos tiene las siguientes propiedades implícitas:

-Representa algún aspecto del mundo real
-Es una colección lógica y coherente de datos con significado inherente.
-Está diseñada, construida y poblada con datos para un fin específico.

Un Sistema Manejador de Bases de Datos (DBMS, Database Manager System) es una colección de programas de software que habilita a los usuarios el crear y mantener una base de datos. Es un sistema de software de uso general que facilita el proceso de definir, construir y manipular bases de datos para varias aplicaciones.

El modelo Relacional

- Fue introducido por el Dr. E. Codd en 1970.
- Está basado en estructuras de datos simples y uniformes, además tiene una sólida base teórica.
- Representa a la base de datos como una colección de relaciones.
- Cuando una relación es considerada como una tabla de valores, cada renglón en la tabla representa una colección de valores de datos relacionados.
- Los nombres de la tabla y de las columnas ayudan a interpretar el significado de los valores en cada renglón.

Todos los valores en una columna son del mismo tipo de datos.
En está terminología, un renglón es llamado un tupla, una columna es llamada un atributo, y la tabla es llamada una relación.
El tipo de dato que describe los tipos de valores que pueden aparecer en cada columna es llamado el dominio.
El dominio es un conjunto de valores atómicos; esto es, cada valor es indivisible.

¿Qué es SQL?

SQL es un sistema basado en un potente lenguaje, para organizar, administrar y consultar datos almacenados en una computadora. SQL es una sigla que deviene de su nombre en inglés "Structured Query Language" (Lenguaje de Consulta Estructurado). Más específicamente SQL esta definido en torno al modelo de bases de datos relacionales, basado en el álgebra relacional, esto le da a SQL las ventajas que lo imponen como el sistema de mayor aceptación.
IBM[6] empezó a comercializar en 1981 el SQL y desde entonces este producto ha tenido un papel importante en el desarrollo de la bases de datos relacionales. IBM propuso y fue

[6] International Business Machines o IBM, conocida coloquialmente como el Gigante Azul, es una empresa que fabrica y comercializa hardware, software y servicios relacionados con la informática.

aceptada, una versión de SQL al Instituto de Estándares Nacional Americano (ANSI) y desde entonces es utilizado de forma generalizada en las bases de datos relacionales.
Algunas de las ventajas de SQL son:

- Marco teórico sólido, fundamentado en el álgebra relacional
- Simplicidad de conceptos (modelo de base de datos: tablas=líneas columnas)
- Definición de vínculos en la consulta, esto le da a SQL una gran flexibilidad
- Fácil y rápido aprendizaje
- Arquitectura cliente-servidor
- Integración con cualquier lenguaje de programación
- Estandarización

¿Que es MySQL?

MySQL es un gestor de bases de datos (SADB). Es una implementación Cliente-Servidor que consta de un servidor y diferentes clientes (programas/librerías).Podemos agregar, acceder, y procesar datos alamacenados en una base de datos.
Es un Sistema de Gestión de Base de Datos Relacional. El modelo relacional se caracteriza a muy grandes rasgos por disponer que toda la información debe estar contenida en tablas, y las relaciones entre datos deben ser representadas explícitamente en esos mismos datos. Esto añade velocidad y flexibilidad.

MySQL es un software de código abierto, ésto quiere decir que es accesible para todos, para usarlo o modificarlo. Podemos descargar MySQL desde Internet y usarlo sin pagar nada, de esta manera cualquiera puede inclinarse a estudiar el código fuente y cambiarlo para adecuarlo a sus necesidades. MySQL está bajo la licencia GPL (GNU Licencia Pública General) para definir qué podemos y qué no podemos hacer con el software en diferentes situaciones.

MySQL es muy rápido, confiable, robusto y fácil de usar tanto para volúmenes de datos grandes como pequeños. Además tiene un conjunto muy práctico de características desarrolladas en cooperación muy cercana con los usuarios. Sin embargo bajo constante desarrollo, MySQL hoy en día ofrece un rico y muy útil conjunto de funciones. La conectividad, velocidad y seguridad hace a MySQL altamente conveniente para acceder a bases de datos en Internet.
Sus principales características son:

- El principal objetivo de MySQL es velocidad y robustez.
- Clientes C, C++, JAVA, Perl, TCL.
- Multiproceso, es decir puede usar varias CPU si éstas están disponibles.
- Puede trabajar en distintas plataformas y S.O. distintos.
- Sistema de contraseñas y privilegios muy flexible y segura.
- Utilidad para chequear, optimizar y reparar tablas.
- El servidor soporta mensajes de error en distintas lenguas.
- Diversos tipos de columnas como enteros de 1, 2, 3, 4, y 8 bytes, coma flotante, doble precisión, carácter, fechas, etc.

INGENIERÍA DE SOFTWARE

Conceptos y objetivos
La ingeniería de software es el establecimiento y uso de principios de ingeniería a fin de obtener software que sea fiable y que funcione eficientemente en un entorno real. Tiene como objetivo servir como base para la producción de software de calidad. La calidad es el grado con que el sistema cumple los requerimientos especificados y necesidades o expectativas del cliente.

Abarca un conjunto de tres elementos importantes:

- Métodos. Indican cómo construir técnicamente el software, incluyen técnicas de modelado (fases).

- Herramientas. Suministran un soporte automático para los métodos, haciendo uso de CASE (Ingeniería de software asistido por computadora).

- Procedimientos. Es la unión que mantiene juntos los métodos y las herramientas y facilita un desarrollo racional y oportuno del software. Definen la secuencia en que se aplican los métodos. Por ejemplo el RUP[7] (Proceso Unificado de Rational), Extreme Programing[8].

Las fases que componen el desarrollo de software son:

-Requisitos

La actividad o modelo de requisitos tiene como meta definir y delimitar la funcionalidad del sistema de software. El modelo de requisitos puede servir como base de negociación y contrato entre el desarrollador del sistema y el cliente, y por lo tanto debe reflejar los deseos del cliente. Es esencial que los clientes que no tengan un conocimiento de computación puedan comprender el modelo de requisitos para facilitar la interacción con ellos.
El modelo de requisitos gobierna el desarrollo de todos los demás modelos, siendo central durante el desarrollo del sistema completo y se estructura mediante el modelo de análisis, se realiza mediante el modelo de diseño, se implementa mediante el modelo de implementación y se prueba mediante el modelo de pruebas. Todos los demás modelos deben verificarse contra el modelo de requisitos. El modelo de requisitos también sirve como base para el desarrollo de las instrucciones operacionales y los manuales, los cuales son descritos desde el punto de vista del usuario.
El desafío de la especificación de requisitos comienza cuando el experto debe comunicar los conceptos, lo cual no es generalmente posible de hacer adecuadamente por medio de

[7] El Proceso Unificado de Rational (RUP, el original inglés *Rational Unified Process*) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.
[8] La programación extrema es una metodología de la ingeniería en software. La punta de la programación extrema se encuentra en las pruebas ya que el desarrollador continúa haciendo el proyecto mientras este no se encuentre en una etapa funcional y aceptable para el cliente.

una simple expresión. Como resultado, se provee explicaciones múltiples, verbales o escritas. El desarrollador pide y captura estas explicaciones y las integra en una representación coherente. La especificación de requisitos es particularmente difícil cuando la información es incompleta, los expertos no pueden articular lo que saben, o no están seguros o incluso son incoherentes sobre su conocimiento o creencias. Una meta importante es minimizar las diferencias entre los espacios de concepto y el modelo de requisitos. Consecuentemente, una de las necesidades principales de cualquier modelo de requisitos es que sea comprensible para cualquier persona.

-Análisis

Después del desarrollo del modelo de requisitos y de haber sido éste aprobado por parte de los usuarios del sistema, se puede iniciar realmente a desarrollar el sistema. Esto comienza con el desarrollo del modelo de análisis que toma como punto de partida la especificación de requisitos y tiene como meta construir una arquitectura capaz de resolver el problema bajo condiciones ideales. Esto significa que se busca desarrollar una estructura lógica del sistema, la cual debe ser estable, robusta, mantenible y extensible. El análisis se enfoca en qué debe hacer el sistema, en lugar de cómo se supone que lo hará. El alcance del modelo de análisis está directamente relacionado con la naturaleza de los conceptos del modelo. En el caso de la tecnología orientada a objetos, se desea: encontrar los objetos, organizar los objetos, describir cómo los objetos interactúan, definir las operaciones de los objetos y definir los objetos internamente.

-Diseño

El propósito del modelo de diseño es extender la arquitectura general de análisis. Este refinamiento se debe a dos razones principales:

- El modelo de análisis no es suficientemente formal por lo cual para poder llegar al código final se debe refinar las estructuras de la arquitectura general.
- Durante el análisis se asume un mundo ideal para el sistema. En la realidad este mundo ideal debe adaptarse al ambiente donde se implementará el sistema.
- Diseño de objetos. El diseño de objetos consiste de decisiones tácticas, tales como la selección de algoritmos y estructura de datos para satisfacer los objetivos de rendimiento y espacio. El modelo de análisis y el diseño de objetos tienen bastante en común, incluyendo los mismos conceptos, técnicas y notaciones.
- Diseño de Sistema. El diseño de sistema define las decisiones estratégicas sobre como se organiza la funcionalidad del sistema en torno al ambiente de implementación.

-Implementación

El modelo de implementación toma el resultado del modelo de diseño para generar el código fuente. Esta traducción debe ser relativamente sencilla y directa, ya que todas las decisiones han sido hechas en las etapas previas. La especialización al lenguaje de programación o base de datos describe cómo traducir los términos usados en el diseño a los términos y propiedades del lenguaje de implementación. Aunque el diseño de objetos es bastante independiente del lenguaje actual, todos los lenguajes tendrán sus especialidades durante la implementación final incluyendo las bases de datos.

En el modelo de implementación, el concepto de rastreabilidad es también muy importante, dado que al leer el código fuente se debe poder rastrear directamente el modelo de diseño y análisis.

- Lenguajes de Programación. Un aspecto importante durante el diseño de objetos es la selección del lenguaje de programación. El uso de un lenguaje de programación orientado a objetos hace más fácil la implementación de un diseño orientado a objetos. La elección del lenguaje influye en el diseño, pero el diseño no debe depender de los detalles del lenguaje. Si se cambia de lenguaje de programación no debe requerirse el re-diseño del sistema. Los lenguajes de programación y sistemas operativos difieren mucho en su organización, aunque la mayoría de los lenguajes tienen la habilidad de expresar los aspectos de la especificación de software que son las estructuras de datos (objetos), flujo dinámico de control secuencial (ciclos, condiciones) o declarativo (reglas, tablas) además de contar con transformaciones funcionales.
- Bases de Datos. Las bases de datos son parte integral de los sistemas de software, en especial de los sistemas de información. En general, se pueden utilizar bases de datos relacionales u otros tipos de bases de datos, como las orientadas a objetos. Si los aspectos dinámicos y funcionales del sistema son menores en relación a las estructuras del sistema, una base de datos relacional puede que sea suficiente ya que éstas se dedican a almacenar principalmente los aspectos estructurales del sistema.

-Integración

El modelo de integración es un aspecto muy importante del proceso de desarrollo. Una característica primordial en todo sistema es mantener la modularidad en los subsistemas, esto significa que inicialmente los subsistemas se desarrollan de manera independiente, llegando el momento para su integración final. Este enfoque maneja de mejor forma la complejidad del sistema y significa que también deberán hacerse pruebas, primero en los componentes por separado y luego en su totalidad.

-Pruebas

El modelo de pruebas es quizás el responsable de revisar la calidad del sistema siendo desarrollado. Los aspectos fundamentales de este modelo son básicamente la prueba de especificación y la prueba de resultado. Probar un sistema es relativamente independiente del método utilizado para desarrollarlo. Las pruebas comienzan con los niveles más bajos, como son los módulos de objetos, hasta llegar a la prueba de integración donde se van integrando partes cada vez más grandes. Una herramienta para pruebas de integración involucra usar el modelo de requisitos para integrar requisitos de manera incremental. En particular, las actividades de pruebas normalmente se dividen en verificación y validación.

- Verificación: Verificación prueba si los resultados están conformes a la especificación, en otras palabras si se está construyendo el sistema correctamente. La verificación debe comenzar lo antes posible, desde el nivel más bajo mediante la verificación de subsistemas, progresando hacia la verificación de la integración, donde las unidades se verifican juntas para ver si interactúan de forma correcta. Finalmente se verifica el sistema completo. La etapa de verificación en la orientación a objetos es menos dramática que en los sistemas que separan funciones de datos, ya que los objetos son unidades más grandes y durante su diseño las unidades ya se están verificando. Por otro lado, herencia al

igual que polimorfismo pueden dificultar la verificación, ya que las operaciones varían según los ancestros o descendientes y los datos de verificación deben ser elegidos cuidadosamente. Incluso la especificación de verificación puede considerarse como una extensión al modelo de requisitos y ser integrada en la arquitectura del sistema. La especificación de verificación debe aplicarse a todos los modelos descritos.

- Validación: Validación prueba si los resultados corresponden a lo que el cliente realmente quería. Este enfoque en la satisfacción del cliente se concentra en obtener la especificación y el resultado correcto. Se hace la pregunta de si se está construyendo el sistema "correcto", en contraste a la verificación donde se pregunta si se está haciendo el sistema "correctamente". La validación se captura por medio de análisis extensivo del modelo de requisitos incluyendo interacción constante con los clientes mediante, uso de prototipos, etc. Se debe validar los resultados del análisis. Según el sistema crece y se formaliza, se estudia qué tan bien el modelo de análisis y el modelo de requisitos describen al sistema. Durante el diseño, se puede ya ir viendo si los resultados del análisis son apropiados para el diseño.

-Documentación

La documentación debe ocurrir a lo largo del desarrollo del sistema y no como una etapa final del mismo. Existen diferentes tipos de documentos que deben ser generados como apoyo al sistema. Cada uno de estos documentos tiene diferentes objetivos y está dirigidos a distintos tipos de personas, desde los usuarios no técnicos hasta los desarrolladores más técnicos. Los siguientes son algunos de los documentos o manuales más importantes:

- Manual del Usuario, que le permite a un usuario comprender como utilizar el sistema.
- Manual del Programador, que le permite a un desarrollador entender los aspectos de diseño considerados durante su implementación.
- Manual del Operador, que le permite al encargado de operar el sistema comprender que pasos debe llevar a cabo para que el sistema funcione bajo cierta configuración de ambiente particular.
- Manual del Administrador, que le permite al encargado de administrar el sistema comprender aspectos más generales como son los modelos de requisitos y análisis.

Realmente no hay límite al número y detalle que se puede lograr mediante la documentación, de manera similar a que no hay límite a qué tanto se puede extender y optimizar un sistema. La idea básica es mantener un nivel de documentación que sea útil aunque es necesario adaptarlo al proceso de la organización.

-Mantenimiento

Una visión equivocada del mantenimiento de un sistema es que involucra únicamente la corrección de errores. El mantenimiento realmente va más allá de corregir problemas y debe basarse principalmente en considerar las extensiones al sistema según nuevas necesidades. En otras palabras, se basa en generar nuevos desarrollos pero tomando como punto de partida el sistema ya existente. En cierta manera es regresar al resto de las actividades pero sin partir de cero. El proceso de mantenimiento tendrá que adaptarse a las nuevas circunstancias del desarrollo.

Patrón de Diseño Modelo/Vista/Controlador (MVC)

Es un patrón de diseño aportado originalmente por el lenguaje SmallTalk[9] a la Ingeniería del Software, se usa comúnmente en sistemas Web. Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales son que el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas.

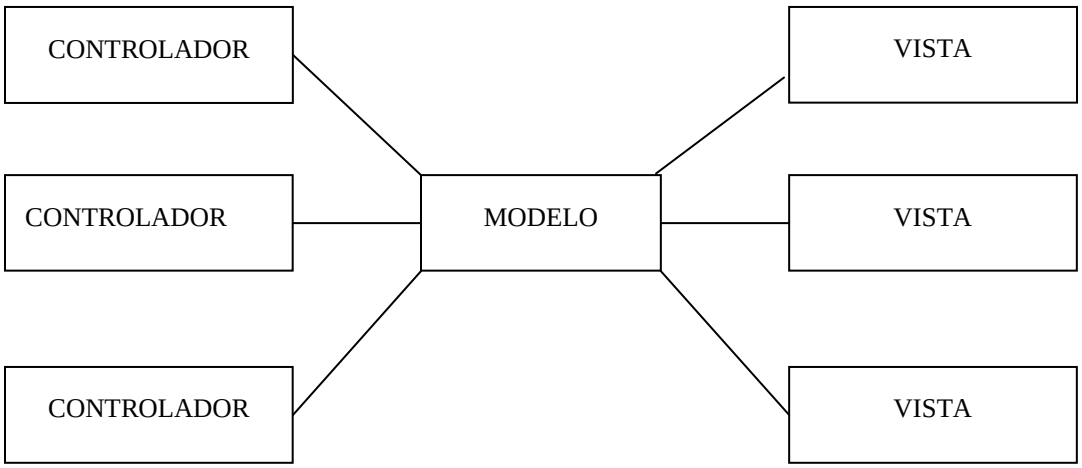


Figura 1.1. La arquitectura MVC en su forma más general. Hay un Modelo, múltiples Controladores que manipulan ese Modelo, y hay varias Vistas de los datos del Modelo, que cambian cuando cambia el estado de ese Modelo.

Este modelo de arquitectura presenta varias ventajas:

- Hay una clara separación entre los componentes de un programa.
- Hay un API[10] muy bien definido; cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.
- La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.

[9] Smalltalk es considerado el primero de los lenguajes de programación orientados a objetos.

[10] API: Una API (del inglés Application Programming Interface - Interfaz de Programación de Aplicaciones) es un conjunto de especificaciones de comunicación entre componentes software. Uno de sus principales propósitos consiste en proporcionar un conjunto de funciones de uso general. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio.

Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Si uno de los Componentes, posteriormente, se observa que funciona mal, puede reemplazarse sin que las otras piezas se vean afectadas.

El **Modelo** es el componente que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo.

La **Vista** es el componente que maneja el aspecto visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo.

El **Controlador** es el componente que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo.

Diferencias del modelo MVC con los modelos convencionales.

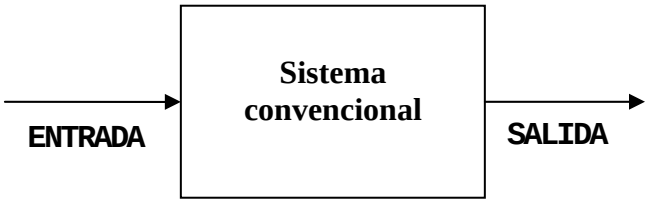


Figura 1.2. Esquema general del modelo convencional.

En el caso del patrón MVC el procesamiento se lleva a cabo entre sus tres componentes. El controlador recibe una orden y decide quien la lleva a cabo en el modelo.

Una vez que el modelo (la lógica de negocio) termina sus operaciones devuelve el flujo vuelve al controlador y este envía el resultado a la capa de presentación.

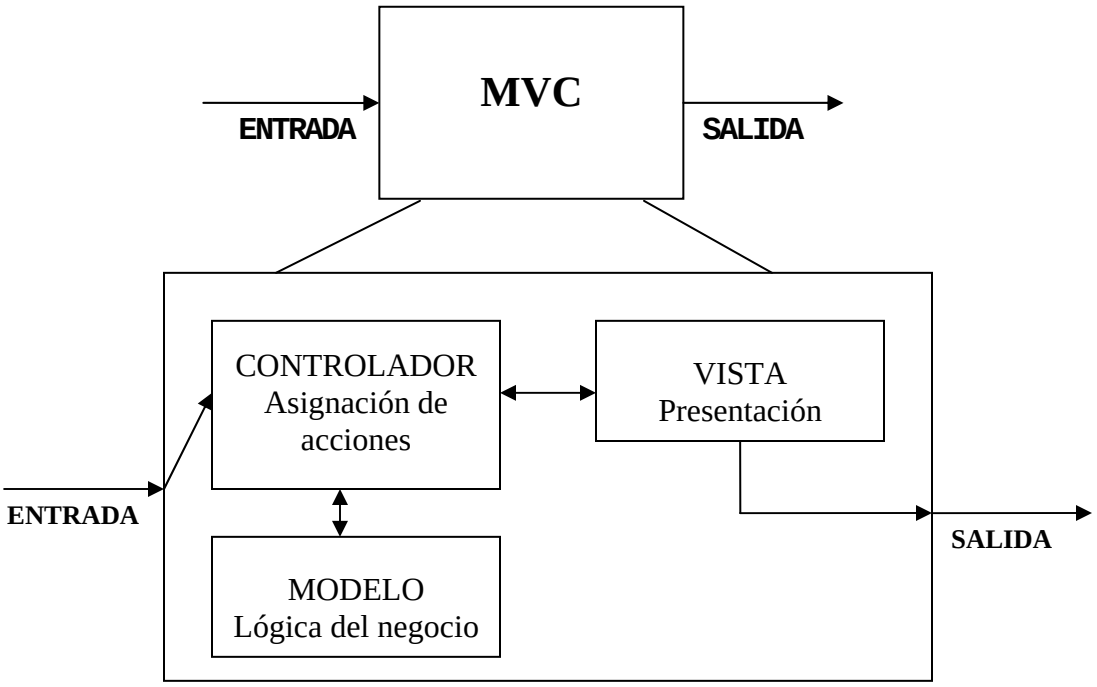


Figura 1.3. Esquema general del Modelo Vista Controlador.

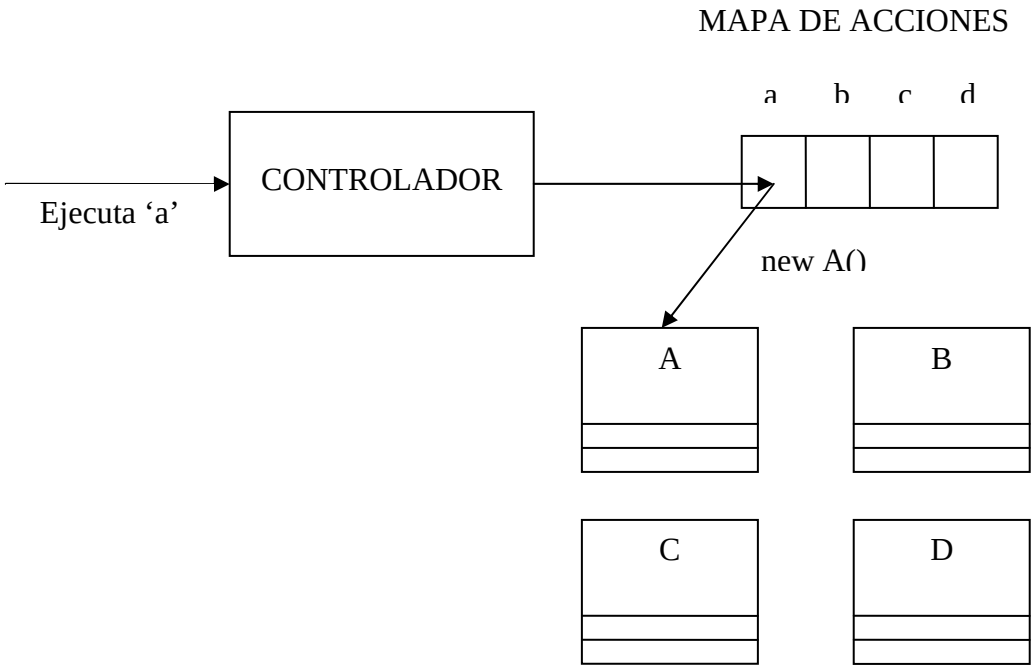


Figura 1.4. El Controlador en cierta forma debe tener un registro de la relación entre órdenes que le pueden llegar y la lógica de negocio que le corresponde.

Otras ventajas que obtenemos de este modelo es una separación total entre lógica de negocio y presentación. A esto se le pueden aplicar opciones como el multilenguaje, distintos diseños de presentación, etc. sin alterar la lógica de negocio.
La separación de capas como presentación, lógica de negocio, acceso a datos es fundamental para el desarrollo de arquitecturas consistentes, reutilizables y más fácilmente mantenibles, lo que al final resulta en un ahorro de tiempo en desarrollo en posteriores proyectos.

El lenguaje Java dispone de unas clases muy sencillas para implantar el modelo MVC: la clase Observer, Observable del paquete util.
Aunque esa implementación del MVC con esas clases se podría hacer a un nivel muy simple de interacción entre unas pocas clases, mediante Struts (framework para aplicaciones web java) se aplica el MVC en toda una aplicación Web convencional.

OTROS PATRONES DE DISEÑO

El patrón de acceso a datos (DAO)

El propósito del patrón DAO es, en pocas palabras, abstraer y encapsular todos los accesos a la fuente de datos. Los objetos de negocio requieren acceder a datos. Estos datos pueden estar almacenados de modos distintos. A la lógica de negocio le debería dar igual como estos datos estén almacenados y como se realicen los accesos.

Con esto se obtiene varias ventajas:

- Se tiene un paliativo al problema del diferencial de impedancia (transparencia);
- Se baja en nivel de acoplamiento entre clases, reduciendo la complejidad de realizar cambios; y
- Se aisla las conexiones a la fuente de datos en una capa fácilmente identificable y mantenible.

El Patrón Value Object

Desde un punto de vista simple, cuando invocamos un método de clase (específicamente en java), podemos pasar los parámetros agrupados en un VO (Value Object)... incluso los valores en la sesión de un servlet, podría agruparse en un Value Object, que contiene todos los atributos encapsulados con sus respectivos métodos accesores; básicamente un VO es un bean (un modelo de componente portable, en el cuarto capítulo se dan más detalles).

Un mecanismo sencillo, en multitud de aplicaciones, es retornar un objeto simple (Value Object). De este modo, el rendimiento del sistema se ve mejorado.

Aunque no debemos abusar. Este abuso se suele llamar Golden Hammer o martillo de oro que consiste en, una vez conocida una técnica, aplicarla indiscriminadamente en todas las situaciones (aún sin hacer realmente falta).

II. ANÁLISIS Y DISEÑO CON UML

Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

El punto importante para notar aquí es que UML es un "lenguaje" para especificar y no un método o un proceso. UML se usa para definir un sistema de software; para detallar los artefactos en el sistema; para documentar y construir -es el lenguaje en el que está descrito el modelo. UML se puede usar en una gran variedad de formas para soportar una metodología de desarrollo de software pero no especifica en sí mismo qué metodología o proceso usar.

Críticas a UML
A pesar de su status de estándar ampliamente reconocido y utilizado, UML siempre ha sido muy criticado por su carencia de una semántica precisa, lo que ha dado lugar a que la interpretación de un modelo UML no pueda ser objetiva. Otro problema de UML es que no se presta con facilidad al diseño de sistemas distribuidos. En tales sistemas cobran importancia factores como transmisión, serialización, persistencia, etc. UML no cuenta con maneras de describir tales factores. No se puede, por ejemplo, usar UML para señalar que un objeto es persistente, o remoto, o que existe en un servidor que corre continuamente y que es compartido entre varias instancias de ejecución del sistema analizado.

Diagramas

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas. Para comprenderlos, a veces es útil categorizarlos jerárquicamente:
Diagramas de estructura enfatizan en los elementos que deben existir en el sistema modelado:

- Diagrama de clases
- Diagrama de objetos
- Diagrama de paquetes

Diagramas de comportamiento enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de actividades
- Diagrama de casos de uso
- Diagrama de estados

Diagramas de Interacción, un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de secuencia
- Diagrama de comunicación

Diagrama de clases

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenimiento.
Elementos

- Clase. Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).
  En UML, una clase es representada por un rectángulo que posee tres divisiones.

  En donde:
  - Superior: Contiene el nombre de la Clase
  - Intermedio: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).
  - Inferior: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

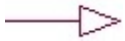
  Atributos: Los atributos o características de una Clase pueden ser de cuatro tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

  - public (+, ): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accsesible desde todos lados.
  - private (-, ): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden accesar).
  - protected (#, ): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de las subclases que se deriven (ver herencia más adelante).
  - friendly: Indica que el atributo sólo es visible dentro del paquete que incluye la clase que lo definió.

  Métodos: Estos pueden tener las características:
  - public (+): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
  - private (-): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden accesar).
  - protected (#): Indica que el método no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de métodos de las subclases que se deriven.
  - friendly: Indica que el método sólo es visible dentro del paquete que incluye la clase que lo definió.

- Relaciones entre Clases. Ahora ya definido el concepto de Clase, es necesario explicar como se pueden interrelacionar dos o más clases.

  En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:
  - uno o muchos: 1..* (1..n)
  - 0 o muchos: 0..* (0..n)
  - número fijo: m (m denota el número).

Herencia (Especialización/Generalización): 

Indica que una subclase hereda los métodos y atributos especificados por una Super Clase, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Super Clase (public y protected).

Agregación: 

Para modelar objetos complejos, no bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, tenemos dos posibilidades:

- Por Valor: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada Composición (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").
- Por Referencia: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada Agregación (el objeto base utiliza al incluido para su funcionamiento).

Asociación: 

Permite asociar objetos que colaboran entre si. El tiempo de vida de un objeto no depende del otro.

Diagrama de Casos de uso

Un diagrama de Casos de Uso muestra las distintas operaciones que se esperan de una aplicación o sistema y cómo se relaciona con su entorno (usuarios u otras aplicaciones).

Caso de uso
Se representa en el diagrama por una elipse, denota un requerimiento solucionado por el sistema. Cada caso de uso es una operación completa desarrollada por los actores y por el sistema en un diálogo. El conjunto de casos de uso representa la totalidad de operaciones desarrolladas por el sistema. Va acompañado de un nombre significativo.

Actor
Es un usuario del sistema, que necesita o usa algunos de los casos de uso. Se representa mediante un símbolo de un “hombrecillo”, acompañado de un nombre significativo, si es necesario.

Relaciones en un diagrama de casos de uso
Entre los elementos de un diagrama de Casos de uso se pueden presentar tres tipos de relaciones, representadas por líneas dirigidas entre ellos (del elemento dependiente al independiente) :

- Asociación. Relación entre un actor y un caso de uso, denota la participación del actor en el caso de uso determinado.
- Usa (uses). Relación entre dos casos de uso, denota la inclusión del comportamiento de un escenario en otro.
- Extiende (extends). Relación entre dos casos de uso, denota cuando un caso de uso es una especialización de otro. Por ejemplo, podría tenerse un caso de uso que extienda la forma de pedir azúcar, que permita escoger el tipo de azúcar (normal, dietética, morena) y además la cantidad en las unidades adecuadas para cada caso (cucharaditas, bolsitas o cucharaditas, respectivamente).

Diagrama de estados

Muestra el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación, junto con los cambios que permiten pasar de un estado a otro.

Estado
Identifica un periodo de tiempo del objeto (no instantáneo) en el cual el objeto esta esperando alguna operación, tiene cierto estado característico o puede recibir cierto tipo de estímulos. Se representa mediante un rectángulo con los bordes redondeados, que puede tener tres compartimientos: uno para el nombre, otro para el valor característico de los atributos del objeto en ese estado y otro para las acciones que se realizan al entrar, salir o estar en un estado (entry, exit o do, respectivamente).

Eventos
Es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Esta ocurrencia puede ser una de varias cosas:

- Condición que toma el valor de verdadero o falso
- Recepción de una señal de otro objeto en el modelo
- Recepción de un mensaje
- Paso de cierto período de tiempo, después de entrar al estado o de cierta hora y fecha particular

El nombre de un evento tiene alcance dentro del paquete en el cual está definido, no es local a la clase que lo nombre.

Envío de mensajes
Además de mostrar y transición de estados por medio de eventos, puede representarse el momento en el cual se envían mensajes a otros objetos. Esto se realiza mediante una línea punteada dirigida al diagrama de estados del objeto receptor del mensaje.

Transición simple
Una transición simple es una relación entre dos estados que indica que un objeto en el primer estado puede entrar al segundo estado y ejecutar ciertas operaciones, cuando un evento ocurre y si ciertas condiciones son satisfechas. Se representa como una línea sólida entre dos estados.

Transición interna
Es una transición que permanece en el mismo estado, en vez de involucrar dos estados distintos. Representa un evento que no causa cambio de estado. Se denota como una cadena adicional en el compartimiento de acciones del estado.
Supongamos el estado de una interfaz pidiendo password al usuario. En este caso puede tenerse una transición interna que muestre una ayuda al usuario.

Diagrama de actividad

Un diagrama de actividades es un caso especial de un diagrama de estados en el cual casi todos los estados son estados de acción (identifican que acción se ejecuta al estar en él) y casi todas las transiciones son enviadas al terminar la acción ejecutada en el estado anterior. Puede dar detalle a un caso de uso, un objeto o un mensaje en un objeto. Sirven para representar transiciones internas, sin hacer mucho énfasis en transiciones o eventos externos. Generalmente modelan los pasos de un algoritmo.

Estado de acción
Representa un estado con acción interna, con por lo menos una transición que identifica la culminación de la acción (por medio de un evento implícito). No deben tener transiciones internas ni transiciones basadas en eventos (Si este es el caso, represéntelo en un diagrama de estados). Permite modelar un paso dentro del algoritmo. Se representan por un rectángulo con bordes redondeados.

Transiciones
Las flechas entre estados representan transiciones con evento implícito. Pueden tener una condición en el caso de decisiones.
Decisiones
Se representa mediante una transición múltiple que sale de un estado, donde cada camino tiene una etiqueta distinta. Se representa mediante un diamante al cual llega la transición del estado inicial y del cual salen las múltiples transiciones de los estados finales.

ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

Descripción del Problema

La descripción del problema es una exposición muy preliminar de necesidades que sirve únicamente como punto de inicio para comprender los requisitos del sistema. Se trata aquí de simular una descripción preparada por un cliente la cual debe evolucionar por medio del modelo de requisitos para lograr la especificación final del sistema a desarrollarse. La descripción del problema debe ser una descripción de necesidades y no una propuesta para una solución. La descripción inicial puede ser incompleta e informal. En nuestro caso la descripción del problema sería:

SISTEMA WEB PARA UN VIDEOCLUB

El videoclub "Aries" se encarga de proveer el servicio de renta de películas en formato VHS y DVD al público en general. Debido a su crecimiento se ha solicitado un sistema que facilite las labores de los encargados de proveer dicho servicio, y que dicho sistema pueda ser utilizado en las demás sucursales de la empresa.

Las actividades que realizan los encargados y que se desean automatizar son renta y devolución de películas, dar de alta y baja socios, generar credenciales de socio, renovar credenciales, agregar nuevas películas al catálogo y eliminarlas de ser necesario.
El sistema debe mantener la seguridad y disponibilidad de la información de los socios, películas y rentas.

Requerimientos no funcionales. Su solicitud y realización es independiente del funcionamiento del sistema.

1. Requisitos de vista y sentido, "look and feel". El producto debe ser fácil de usar en todos los exploradores y pantallas concebibles desde una pantalla mínima. La estructura debe ser sencilla y fácil de navegar.

2. Requisitos de usabilidad. Facilidades de uso. El sistema debe involucrar pocos pasos. Facilidad de aprendizaje. El sistema debe ser intuitivo y no requerir ningún entrenamiento antes de poder usarse.

3. Requisito de desempeño. El tiempo máximo de espera para una respuesta debe ser de 2 segundos. Las conexiones en Internet deben ser seguras (encriptados).

4. Requisitos operacionales. Hardware mínimo necesario para que el sistema funcione adecuadamente, por ejemplo el tipo de procesador, conexión a red, memoria RAM, etc. También se incluye el software necesario como el tipo de navegador, software de desarrollo.

5. Requisitos de mantenimiento y portabilidad.
6. Requisitos de seguridad
7. Requisitos culturales y políticos
8. Requisitos legales

Requerimientos funcionales. Involucran características relacionadas con la operación del sistema. Es decir los requerimientos que el usuario espera del sistema para realizar su trabajo.

Existen diversos métodos para el levantamiento de los requerimientos del sistema, es decir la obtención de ellos.

Sombra. Observar la actividad del usuario para realizar un análisis de la tarea que este realiza y responder a las preguntas ¿Cómo realiza su trabajo? ¿Qué decisiones toma cuando inicia o termina su trabajo? ¿Con cuántas personas interactúa para realizar una actividad dada? ¿Existen variaciones en los pasos que debe seguir?.

Entrevista. Formulación de preguntas directas, como por ejemplo ¿Qué problemas encuentra al realizar su tarea? ¿Tiene necesidades especiales que no estén documentadas? ¿Qué otros individuos o sistemas afectan su trabajo?.

Enfoque a grupos. Realización de entrevistas pero en grupo donde la respuesta se obtiene a través de un consenso.

Prototipo. Consiste en presentar propuestas que modelen la realización de actividades a través de un prototipo del sistema.

REQUERIMIENTOS DEL SISTEMA VIDEOCLUB

FUNCIONALES DESEABLES

- Dar de alta un socio, requerirá de ingresar los siguientes datos: Fecha de alta, Nombre completo, Clave de elector (IFE), Domicilio, Teléfono, Correo electrónico.

- El sistema debe generar e imprimir las credenciales de socio. Esta contará con un número de folio, fecha de emisión, nombre del socio, fecha de vencimiento, además tendrá vigencia de un año, con capacidad de renovarse por el mismo periodo. Para obtener una credencial de socio, se requiere darse de alta en el sistema, presentar documentación (credencial IFE, comprobante de domicilio).

- Dar de baja a un socio.

- Realizar una renta. Se requiere la credencial de socio vigente, identificar cada película rentada y registrar la fecha de renta. Se prestará un máximo de 5 películas por cada socio. Cada película rentada se devolverá al siguiente día antes del cierre. Por otro lado, el sistema verificará antes de efectuar la renta que el socio no tenga alguna devolución pendiente con retraso.

- Registrar devolución, para ello se requiere recibir la película. Por cada día de retraso en la devolución de una película, se cobrará una renta normal. El sistema debe verificar que cada película que se devuelve no tenga retraso, si así fuera el sistema informará el monto del cargo y dar la opción para registrar el pago.

- Dar de alta y de baja películas, para dar de alta una película se deberán registrar sus características principales: Título, Género, Formato, Clasificación, Precio.

- El sistema debe permitir exclusivamente al encargado (controlando su acceso mediante un usuario y un password) realizar todas las actividades propias del negocio ya mencionadas, incluyendo la edición de la información de socios y películas.

NO FUNCIONALES

- El sistema debe ser fácil de manejar por una persona con los conocimientos mínimos de computación.
- Las pantallas del sistema deben tener como base los colores del logotipo del videoclub.

- PC Pentium IV como mínimo
- Memoria RAM 128 MB como mínimo

- Sistema Operativo: Windows XP
- Herramienta de Desarrollo: Netbeans (Lenguaje Java)
- Manejador de Bases de Datos: MySQL
- Servidor Web: Apache
- Servidor de Aplicaciones: Tomcat
- Conexión a Internet

Modelo de Análisis

El objetivo del modelo de análisis es comprender y generar una arquitectura de objetos para el sistema en base a lo especificado en el modelo de requisitos. Durante esta etapa no se considera el ambiente de implementación (lenguaje de programación, manejador de base de datos, etc.).

El análisis pretende modelar el sistema bajo condiciones ideales, garantizando que la arquitectura de software resultante se suficientemente robusta y extensible para servir de base a la estructura lógica de la aplicación.

Durante el diseño todas las consideraciones que han sido descartadas durante el análisis sean consideradas.

Casos de Uso

Los *casos de uso* describen un sistema en término de sus distintas formas de utilización, cada uno de estas formas es conocida como un *caso de uso*. Cada caso de uso o flujo se compone de una secuencia de eventos iniciada por el usuario.

Dado que los casos de uso describen el sistema a desarrollarse, cambios en los requisitos significarán cambios en los casos de uso. Por ejemplo, un caso de uso para manejar un automóvil sería la secuencia de eventos desde que el conductor entra en el coche encendiendo el motor hasta llegar a su destino final.

Por lo tanto, para comprender los casos de uso de un sistema primero es necesario saber quienes son sus usuarios. Por ejemplo, conducir un automóvil es distinto a arreglarlo, donde los usuarios también son distintos, el dueño del automóvil y el mecánico, respectivamente. Para ello se define el concepto de *actor*, correspondiente al tipo de usuario que está involucrado en la utilización de un sistema, siendo el actor una entidad externa al propio sistema.

A continuación aparece el diagrama de casos de uso general de nuestro sistema y después la documentación necesaria de cada actividad.

Figura 2.1. Diagrama general de casos de uso del sistema

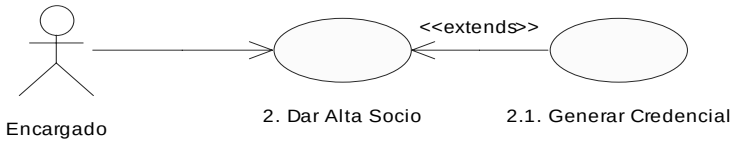
Parte fundamental del modelo de casos de uso es una descripción textual detallada de cada uno de los casos de uso identificados. Esta **documentación** es sumamente críticos ya que a partir de estos se desarrollará el sistema completo.
Las descripciones de los casos de uso representan todas las posibles interacciones de los actores con el sistema únicamente en base a eventos enviados o recibidos por los actores. En esta etapa no se incluyen eventos internos al propio sistema ya que esto será tratado durante el análisis y únicamente agregaría complejidad innecesaria en esta etapa.

DOCUMENTACIÓN CASOS DE USO DEL SISTEMA

CASO DE USO: 1. Ingresar Sistema
ACTOR: Encargado
TIPO: Primario
DESCRIPCIÓN: El encargado ingresa al sistema mediante un usuario y una contraseña.


ACTOR		**SISTEMA**		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Acceder a la página principal del sistema.	2	Mostrar la página de acceso	
3	Captura el usuario y contraseña de acceso. Acepta los datos.	4	Despliega la página principal del sistema.	-E1: Si los datos son inválidos, permite nuevamente la captura. -E2: Si campos vacíos, permite nuevamente la captura.

CASO DE USO: 2. Dar Alta Socio
ACTOR: Encargado
TIPO: Primario
DESCRIPCIÓN: El Encargado puede dar de alta un socio. Previamente debe requerir la documentación (comprobante domicilio, credencial IFE) y después de dar de alta genera la respectiva credencial de socio.



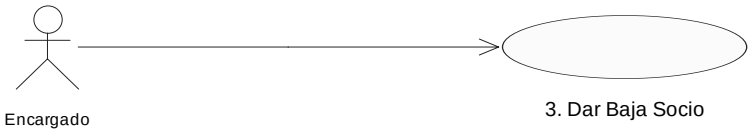
ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de alta socio.	2	Despliega la pantalla correspondiente.	
3	El encargado captura los datos del nuevo socio. Acepta los datos.	4	Efectúa la alta y despliega la pantalla de alta exitosa y la opción de generar credencial.	-E1: Si los datos son inválidos, permite nuevamente la captura. -E2: Si campos vacíos, permite nuevamente la captura.
5	Selecciona la opción de generar credencial.	6	Despliega la pantalla con la credencial del nuevo socio y la opción de imprimir credencial.	
7	Selecciona la opción imprimir credencial.	7	Realiza la impresión. Regresa a la pantalla principal.	

CASO DE USO: 3. Dar Baja Socio

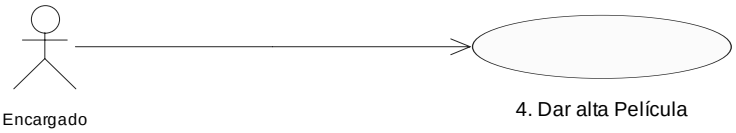
ACTOR: Encargado

TIPO: Secundario

DESCRIPCIÓN: El encargado puede dar de baja del sistema a un socio. Ya sea por solicitud o por que no ha renovado su credencial en un año, etc.



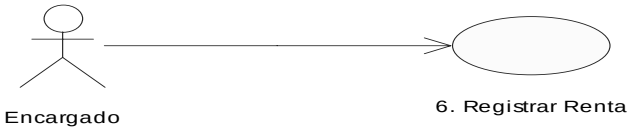
ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de baja socio.	2	Despliega la pantalla correspondiente y solicita el identificador del socio.	
3	El encargado captura el identificador del socio. Acepta el dato.	4	Despliega la pantalla correspondiente con la información del socio.	-E1: Si identificador inválido, permite nuevamente la captura. -E2: Si campo vacío, permite nuevamente la captura.
5	Acepta la baja del socio.	6	Efectúa la baja y despliega la pantalla de baja exitosa y permite regresar a la pantalla principal.	

CASO DE USO: 4. Dar Alta película
ACTOR: Encargado
TIPO: Primario
DESCRIPCIÓN: El encargado puede dar de alta una película en el sistema, ingresando sus características principales.


ACTOR		**SISTEMA**		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de alta película.	2	Despliega la pantalla correspondiente.	
3	El encargado captura los datos de la nueva película. Acepta los datos.	4	Efectúa la alta y despliega la pantalla de alta exitosa con la opción de regresar a la pantalla principal.	-E1: Si los datos son inválidos, permite nuevamente la captura. -E2: Si campos vacíos, permite nuevamente la captura.

CASO DE USO: 5. Dar Baja Película
ACTOR: Encargado
TIPO: Secundario
DESCRIPCIÓN: El encargado puede dar de baja del sistema una película. Eliminando por completo su registro del sistema.

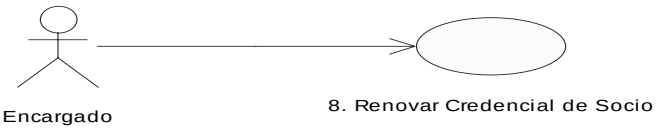

ACTOR		**SISTEMA**		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de baja película.	2	Despliega la pantalla correspondiente y solicita el identificador de la película.	
3	El encargado captura el identificador de la película. Acepta el dato.	4	Despliega la pantalla correspondiente con la información de la película.	-E1: Si identificador inválido, permite nuevamente la captura. -E2: Si campo vacío, permite nuevamente la captura.
5	Acepta la baja de la película.	6	Efectúa la baja y despliega la pantalla de baja exitosa y permite regresar a la pantalla principal.	

CASO DE USO: 6. Registrar Renta
ACTOR: Encargado
TIPO: Primario
DESCRIPCIÓN: El encargado puede registrar la renta de una película. Sólo puede rentar 5 películas por cada socio. No podrá realizar la renta si el socio tiene devolución pendiente con retraso.


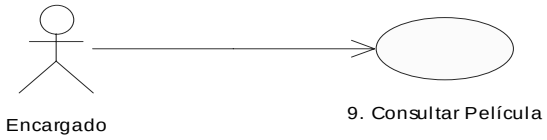
ACTOR		**SISTEMA**		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de registrar renta.	2	Despliega la pantalla correspondiente y solicita el identificador del socio solicitante.	
3	El encargado captura el identificador del socio. Acepta el dato.	4	Despliega la pantalla correspondiente con la información del socio y solicita el identificador de la película.	-E1: Si el identificador es inválido, permite nuevamente la captura. -E2: Si campo vacío, permite nuevamente la captura. -E3: El socio ya no tiene permitida la renta, tiene retraso de devolución o excede el límite de películas (5).
5	Captura el identificador de la película. Acepta el dato	6	Despliega la pantalla correspondiente con la información de la película. Permite la captura de otro identificador, repitiéndose el proceso anterior.	-E4: Si el identificador es inválido, permite nuevamente la captura. -E5: Si campo vacío, permite nuevamente la captura.
7	Corrobora la información relacionada con la renta y la guarda.	7	Guarda la información de la renta. Despliega la pantalla correspondiente de renta exitosa. Permite regresar a la pantalla principal.	

CASO DE USO: 7. Registrar Devolución
ACTOR: Encargado
TIPO: Primario
DESCRIPCIÓN: El encargado puede registrar la devolución de una película rentada previamente. Y deberá efectuar el cobro del cargo por retraso que se pueda generar.


ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de registrar devolución.	2	Despliega la pantalla correspondiente y solicita el identificador de la película.	
3	El encargado captura el identificador de la película. Acepta el dato.	4	Despliega la pantalla correspondiente con la información de la renta que incluye dicha película.	-E1: Si el identificador es inválido, permite nuevamente la captura. -E2: Si campo vacío, permite nuevamente la captura. -E3: El socio tiene retraso de devolución y permite el cobro del cargo.
5	Acepta la devolución.	6	Despliega la pantalla correspondiente de devolución exitosa y permite regresar a la pantalla principal.	

CASO DE USO: 8. Renovar Credencial de Socio
ACTOR: Encargado
TIPO: Primario
DESCRIPCIÓN: El encargado puede renovar la credencial de un socio. Cada credencial tiene una vigencia de un año.


ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de renovar credencial.	2	Despliega la pantalla correspondiente y solicita el identificador del socio.	
3	El encargado captura el identificador del socio. Acepta el dato.	4	Despliega la pantalla correspondiente con los datos de la credencial de socio, y con la posibilidad de modificar la fecha de vigencia.	-E1: Si el identificador es inválido, permite nuevamente la captura. -E2: Si campo vacío, permite nuevamente la captura.
5	Realiza los cambios en la fecha de vigencia de la credencial. Acepta los cambios.	6	Despliega la pantalla correspondiente a la credencial de socio renovada. Permite la impresión de la credencial.	-E3: Fecha inválida, permite nuevamente la captura.
7	Selecciona la opción de imprimir la credencial.	8	Efectúa la impresión y regresa a la pantalla principal.	

CASO DE USO: 9. Consultar Película
ACTOR: Encargado
TIPO: Secundario
DESCRIPCIÓN: El encargado puede realizar la búsqueda de una película en particular y ver la información de ella, además de realizar cambios.


ACTOR		**SISTEMA**		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de consultar película.	2	Despliega la pantalla correspondiente y solicita el identificador de la película.	
3	El encargado captura el identificador de la película. Acepta el dato.	4	Despliega la pantalla correspondiente con la información de la película y con la opción de editar.	-E1: Si identificador inválido, permite nuevamente la captura. -E2: Si campo vacío, permite nuevamente la captura.
5	Selecciona la opción editar.	6	Despliega la pantalla correspondiente donde habilita los datos de la película para permitir su modificación.	
7	Guarda los cambios efectuados.	8	Guarda los cambios y despliega la pantalla correspondiente a la edición exitosa. Permite regresar a la pantalla principal.	

CASO DE USO: 10. Consultar Socio
ACTOR: Encargado
TIPO: Secundario
DESCRIPCIÓN: El encargado puede realizar la búsqueda de un socio en particular y ver la información de él, además de realizar cambios.


ACTOR		**SISTEMA**		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Elige la opción de consultar socio.	2	Despliega la pantalla correspondiente y solicita el identificador del socio.	
3	El encargado captura el identificador del socio. Acepta el dato.	4	Despliega la pantalla correspondiente con la información del socio y con la opción de editar.	-E1: Si identificador inválido, permite nuevamente la captura. -E2: Si campo vacío, permite nuevamente la captura.
5	Selecciona la opción editar.	6	Despliega la pantalla correspondiente donde habilita los datos del socio para permitir su modificación.	
7	Guarda los cambios efectuados.	8	Guarda los cambios y despliega la pantalla correspondiente a la edición exitosa. Permite regresar a la pantalla principal.	

Modelo Conceptual

Es la herramienta más importante del análisis orientado a objetos.
Es un modelo que comunica cuáles son los términos importantes y como se relacionan entre si.
Su finalidad es facilitar el conocimiento del vocabulario del dominio y de los conceptos que se incluyen en los requerimientos.
Es una idea, cosa u objeto; es la representación de las cosas del mundo real y NO de componentes de software.

Cómo encontrar los conceptos
Es necesario analizar los sustantivos en la descripción del sistema, requerimientos y casos de uso.
Aunque no es posible encontrar mecánicamente correspondencias entre sustantivo y concepto.
Uno de los errores más comunes al crear modelos conceptuales, es representar algo como atributo, cuando debería ser un concepto.
Una regla práctica para evitar esto es:

> "Si en el mundo real no consideramos algún elemento X como número o texto, probablemente X sea un concepto no un atributo".

Cuando el número de elementos del modelo empieza a aumentar, se hace necesario buscar una forma de organizarlos y para esto usamos paquetes.
Un paquete UML se muestra gráficamente como una carpeta etiquetada.
Un elemento es propiedad del paquete en el cual está definido, pero puede reverenciarse en otros paquetes.
Si un elemento de un modelo depende de otro, esta dependencia puede mostrarse a través de una relación de subordinación.

Recomendaciones
Crear un paquete llamado Conceptos del dominio. Para agrupar todos los elementos del modelo conceptual.
Los conceptos comunes compartidos por la mayoría de paquetes, se pueden agrupar en un paquete llamado conceptos básicos
Utilizar nombres o vocabulario del dominio.
Excluir características irrelevantes.
No agregar cosas que no existan.
Un modelo conceptual explica los conceptos más significativos en el dominio del problema.
En el no se definen operaciones o métodos; se pueden mostrar los conceptos, los atributos de los conceptos (opcionalmente) y la relación o asociación entre ellos.

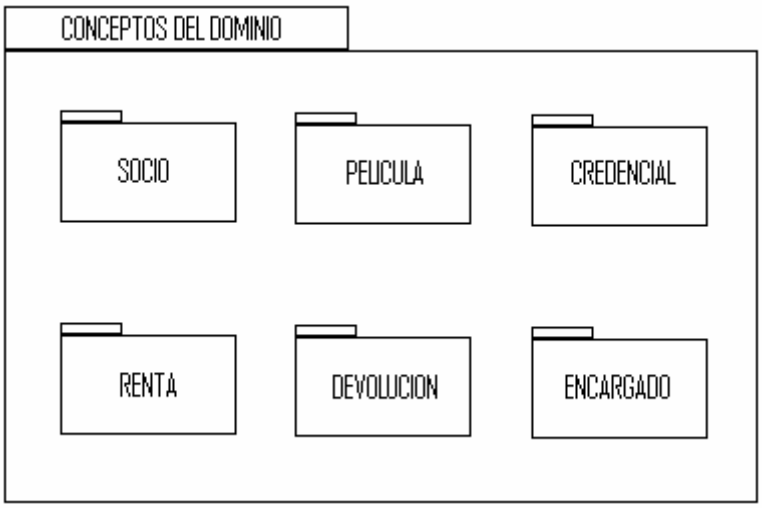


Figura 2.2. Modelo Conceptual del sistema videoclub.

Modelo de Diseño

El modelo de diseño es un refinamiento y formalización adicional del modelo de análisis donde se toman en cuenta las consecuencias del ambiente de implementación. El resultado del modelo de diseño son especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos.

Se requiere un modelo de diseño ya que el modelo de análisis no es lo suficientemente formal para poder llegar al código fuente. Por tal motivo se debe refinar los objetos, incluyendo las operaciones que se deben ofrecer, la comunicación entre los diferentes objetos, los eventos que los objetos envían entre si, etc. El sistema real debe adaptarse al ambiente de implementación. En el análisis se asume un mundo ideal para el sistema, en la realidad se debe adaptar el sistema al ambiente de implementación, algo que puede cambiar durante el ciclo de vida del sistema.

Se busca además aspectos como, los requisitos de rendimiento, necesidades de tiempo real, concurrencia, el lenguaje de programación, el sistema de manejo de base de datos, etc.
Durante el diseño, se puede ver si los resultados del análisis son apropiados para su implementación.

*Diseño de Sistema***.** Se adapta el modelo al ambiente de implementación. Aquí deben ser tomadas las decisiones de implementación estratégicas:

(i) Cómo se incorporará una base de datos en el sistema,
(ii) Qué bibliotecas de componentes se usarán y cómo,
(iii) Qué lenguajes de programación se utilizarán,
(iv) Cómo se manejarán los procesos, incluyendo comunicación y requisitos de rendimiento,

(v) Cómo se diseñará el manejo de excepciones y recolección de basura, etc. En resumen, se debe especificar cómo las circunstancias del ambiente de implementación deben ser manejadas en el sistema.

Diseño de Clases. Se refina y formaliza el modelo para generar especificaciones muy detalladas de todas las clases, incluyendo sus operaciones y atributos. Se describe cómo interaccionan las clases en cada caso de uso específico, especificando qué debe hacer cada operación en cada clase. Este paso genera las interfaces de las clases, las cuales deben ser luego implementadas mediante métodos.

Antes de poder resolver el diseño es necesario tomar decisiones generales sobre las estrategias de diseño a seguir, aspectos que incluyen la *arquitectura*, *robustez*, y *reuso* del sistema.
El término arquitectura se refiere, a la organización de las clases dentro del sistema. La robustez de un sistema debe ser uno de los objetivos principales del diseño. El sistema debe estar protegido contra parámetros incorrectos proporcionados por el usuario. El diseñador de métodos debe considerar dos tipos de condiciones de error:

(i) Errores lógicos que son identificados durante el análisis.
(ii) Errores de implementación, incluyendo errores del sistema operativo, tales como los errores de asignación de memoria, o errores de archivos de entrada y salida, etc.

El encapsulamiento juega un papel fundamental para la robustez del sistema. Ocultar la información interna, atributos e implementación de métodos, a una clase permite que ésta pueda ser cambiada sin afectar al resto del sistema.

A través de la herencia se puede incrementar el reuso de código. Se toman los aspectos comunes a clases similares utilizando superclases comunes.

Arquitectura de Clases

Dependiendo del tipo de aplicación existen diversas arquitecturas que se pueden utilizar.
En el caso de los sistemas de información, uno de las tipos de arquitecturas mas importantes es la arquitectura *MVC-Modelo*, *Vista*, *Control*, conocido también como un patrón de diseño. Esta arquitectura se basa en tres dimensiones principales: *Modelo* (*información*), *Vista* (*presentación*) y *Control* (*comportamiento*). En el modelo de análisis descrito aquí utilizaremos como base la arquitectura *MVC* para capturar estos tres aspectos de la funcionalidad. Es importante notar la correspondencia con las tres dimensiones utilizadas durante el modelo de requisitos.

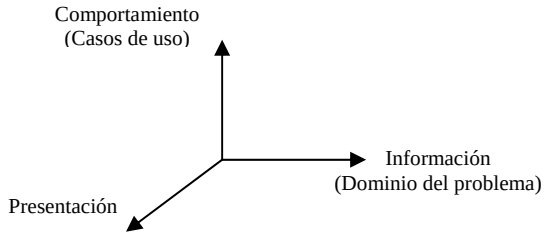


Figura 2.3. Correspondencia con las tres dimensiones utilizadas durante el modelo de requisitos.

La arquitectura se implementa mediante tres tipos o *estereotipos* de objetos como elementos básicos de desarrollo.

Clases con Estereotipos

El tipo de funcionalidad o "la razón de ser" de un objeto dentro de una arquitectura se le conoce como su *estereotipo*. Para los sistemas de información la arquitectura del sistema según nuestro modelo de análisis se basa en tres estereotipos básicos de objetos:

-El estereotipo *entidad* para objetos que guarden información sobre el estado interno del sistema, a corto y largo plazo, correspondiente al dominio del problema.

-El estereotipo *interfase* para objetos que implementen la presentación o vista correspondiente a las interfaces del sistema hacia el mundo externo.

-El estereotipo *control* para objetos que implementen el comportamiento o control especificando cuando y como el sistema cambia de estado, correspondiente a los casos de uso.

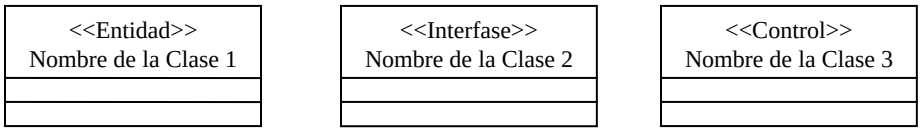


Figura 2.4. Los tres estereotipos correspondientes a las tres dimensiones para la arquitectura del modelo de análisis.

Cuando se trabaja en el desarrollo del modelo de análisis, normalmente se trabaja con un caso de uso a la vez.

Para cada caso de uso se identifican los objetos necesarios para su implementación.

Se debe identificar primero las clases interfase, luego las entidad y finalmente las de control.

DIAGRAMA DE CLASES DEL SISTEMA VIDEOCLUB

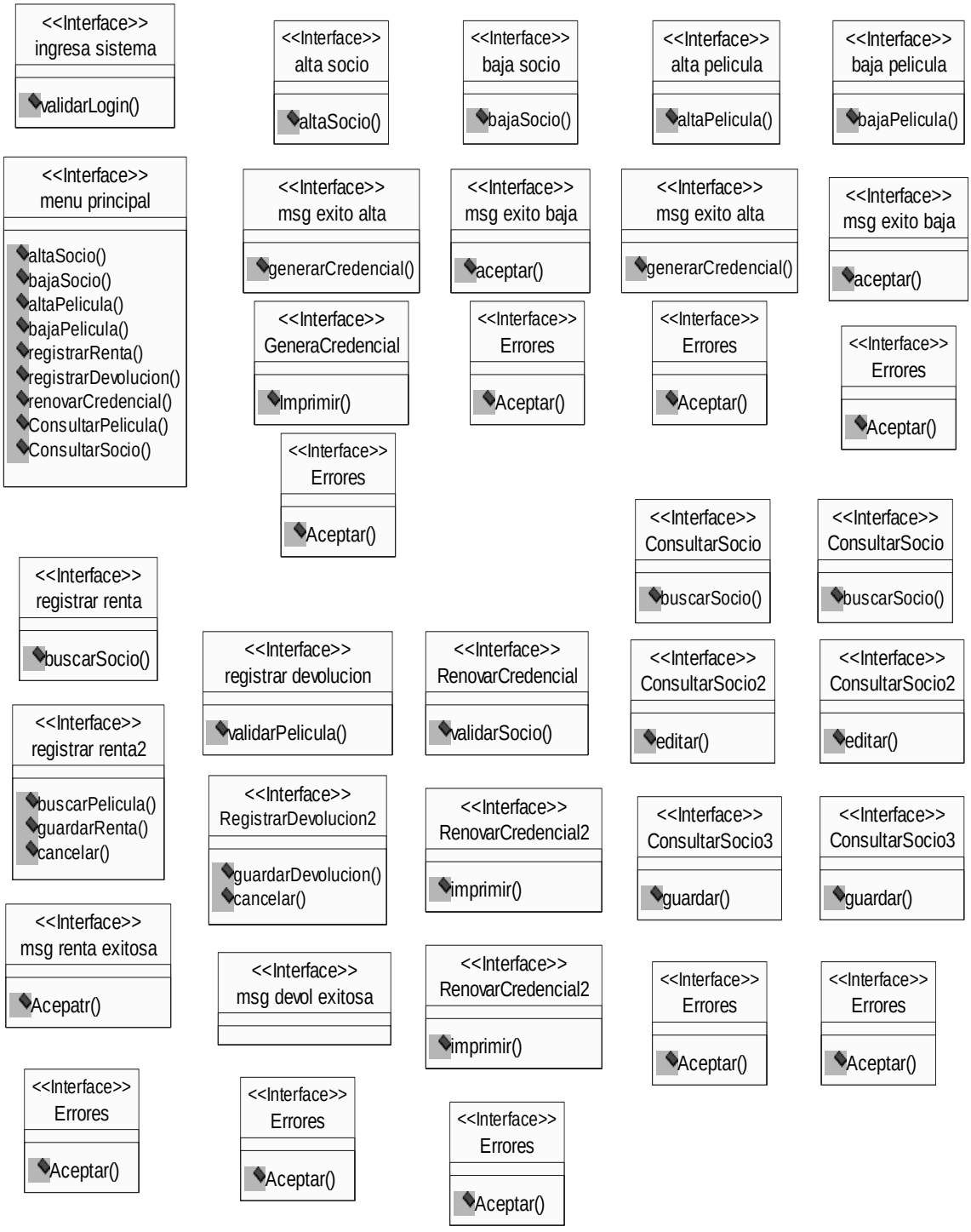


Figura 2.5. Principales clases que representan la interfaz del sistema.

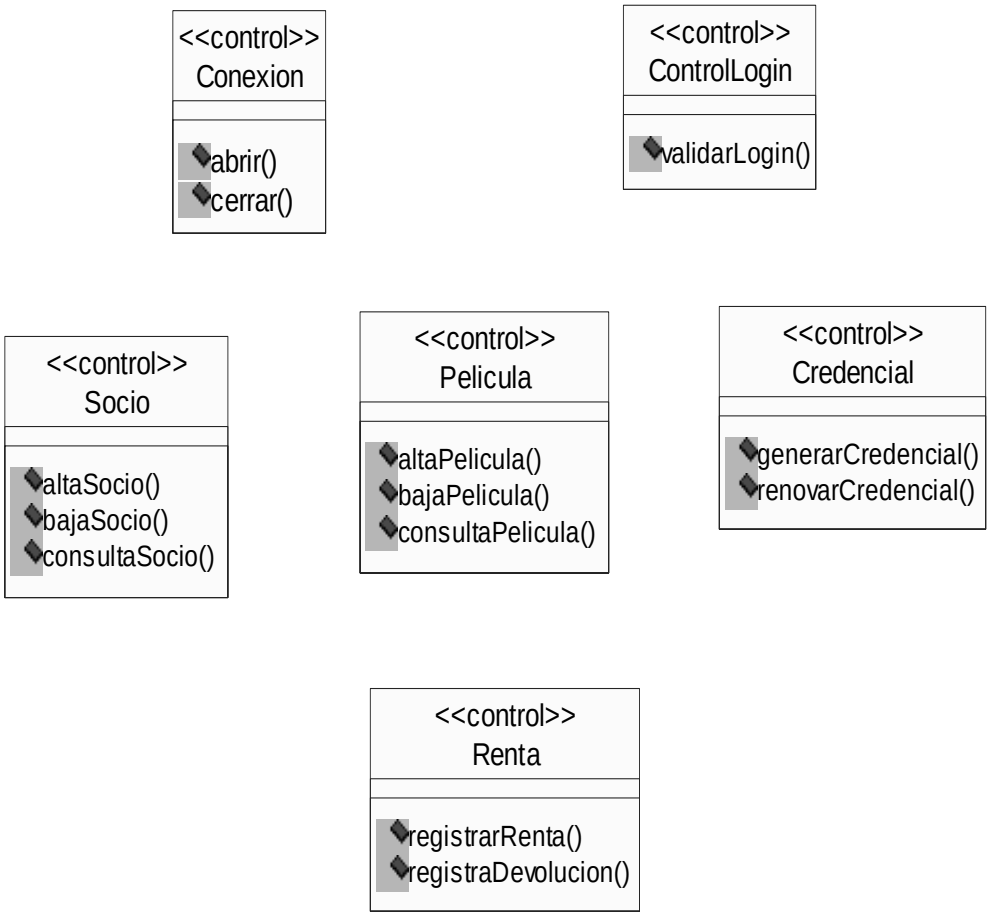


Figura 2.6. Principales clases de control de procesos del sistema.

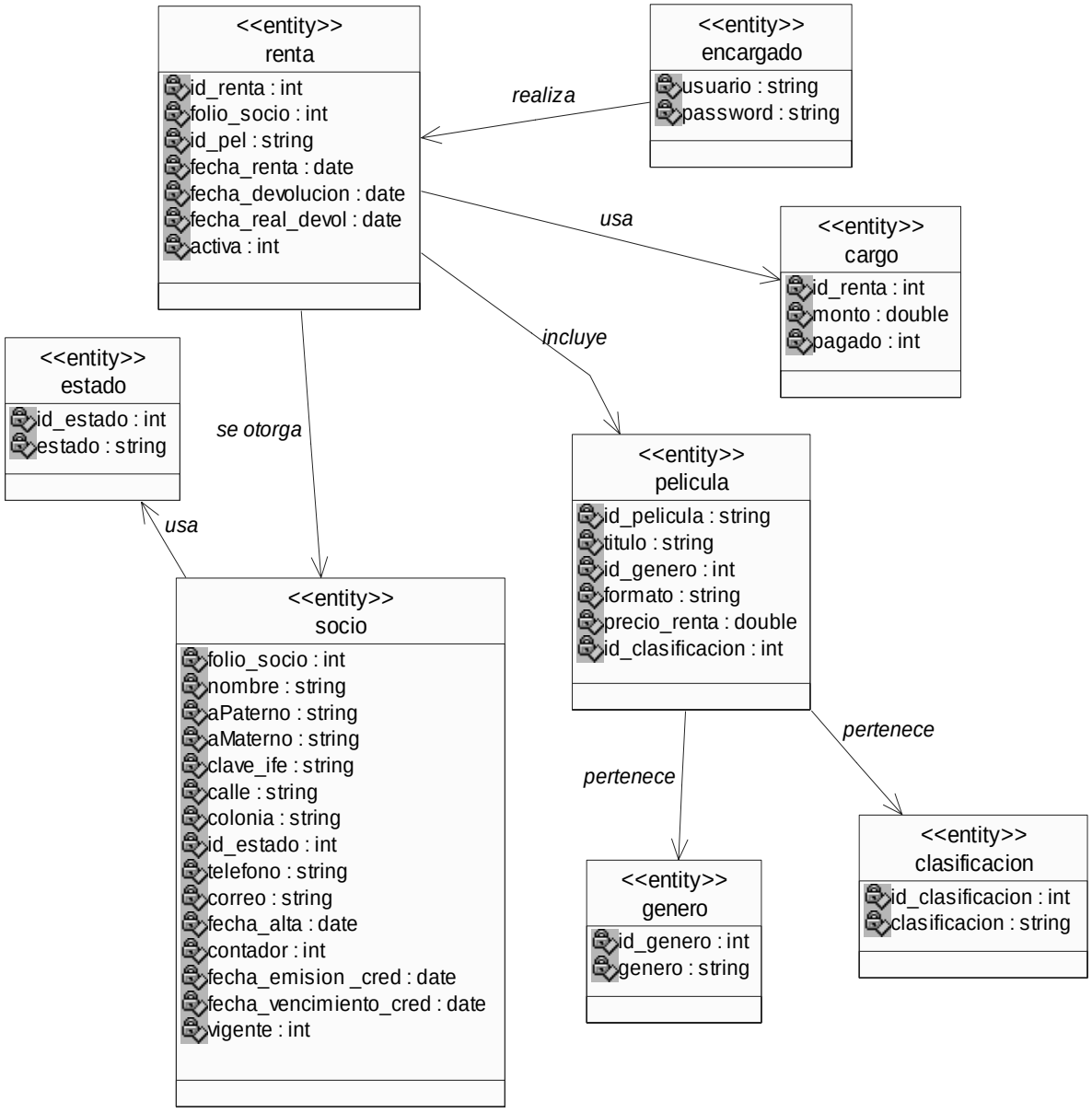


Figura 2.7. Principales clases que representan las entidades o tablas de la base de datos.

DIAGRAMAS DE ACTIVIDADES DEL SISTEMA

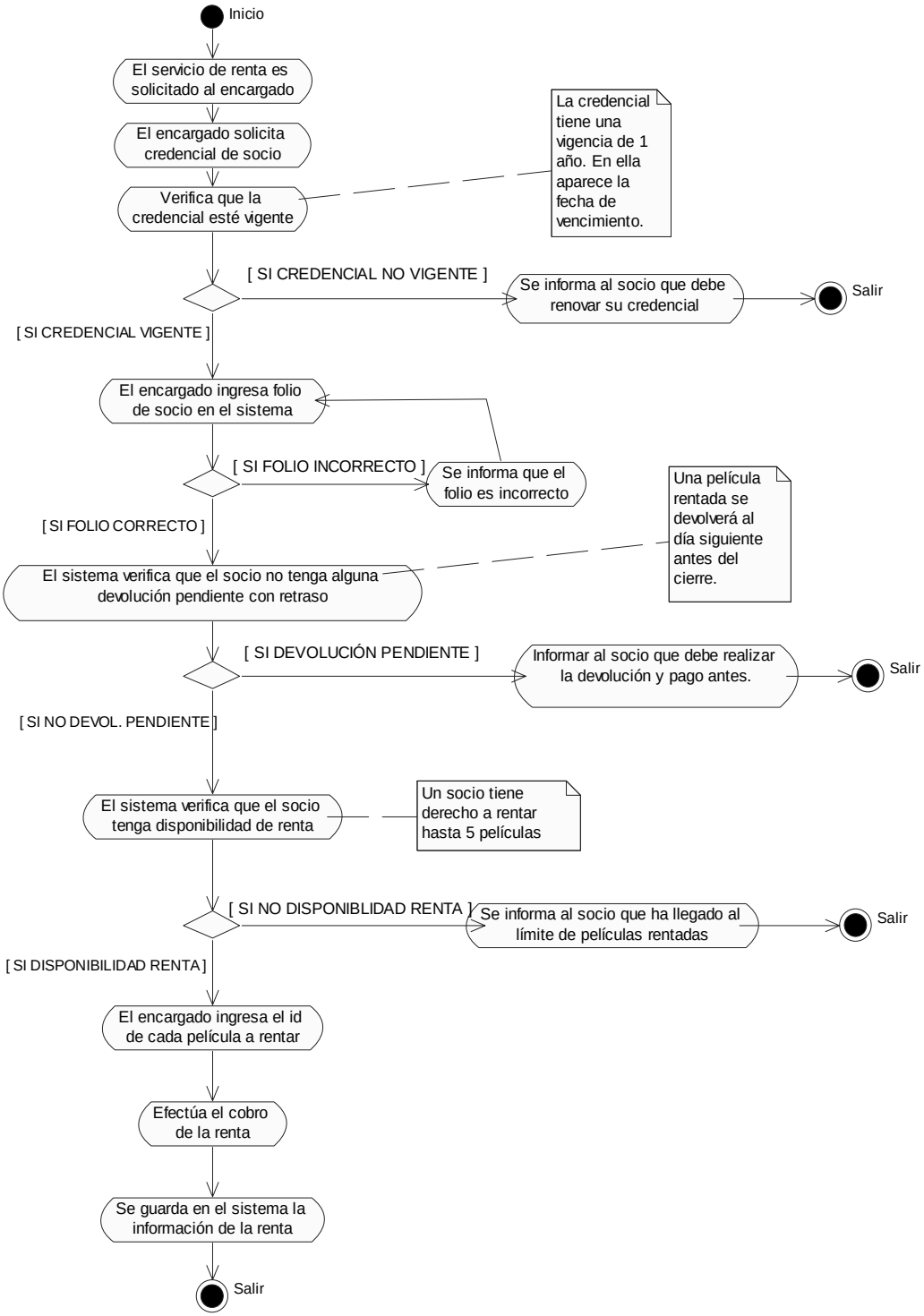


Figura 2.8. Secuencia de pasos seguidos por el encargado para registrar una renta.

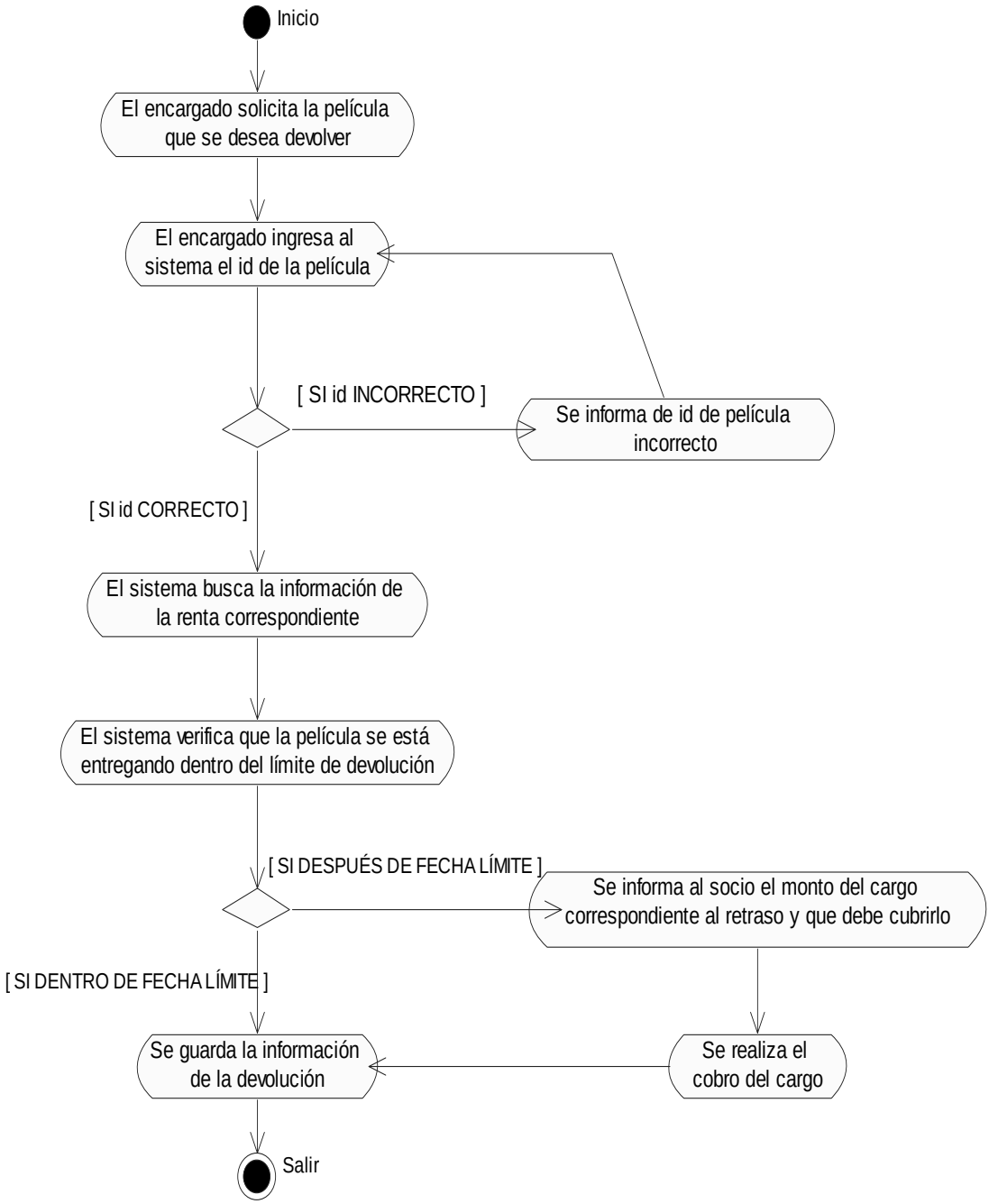


Figura 2.9. Secuencia de pasos efectuados para registrar una devolución.

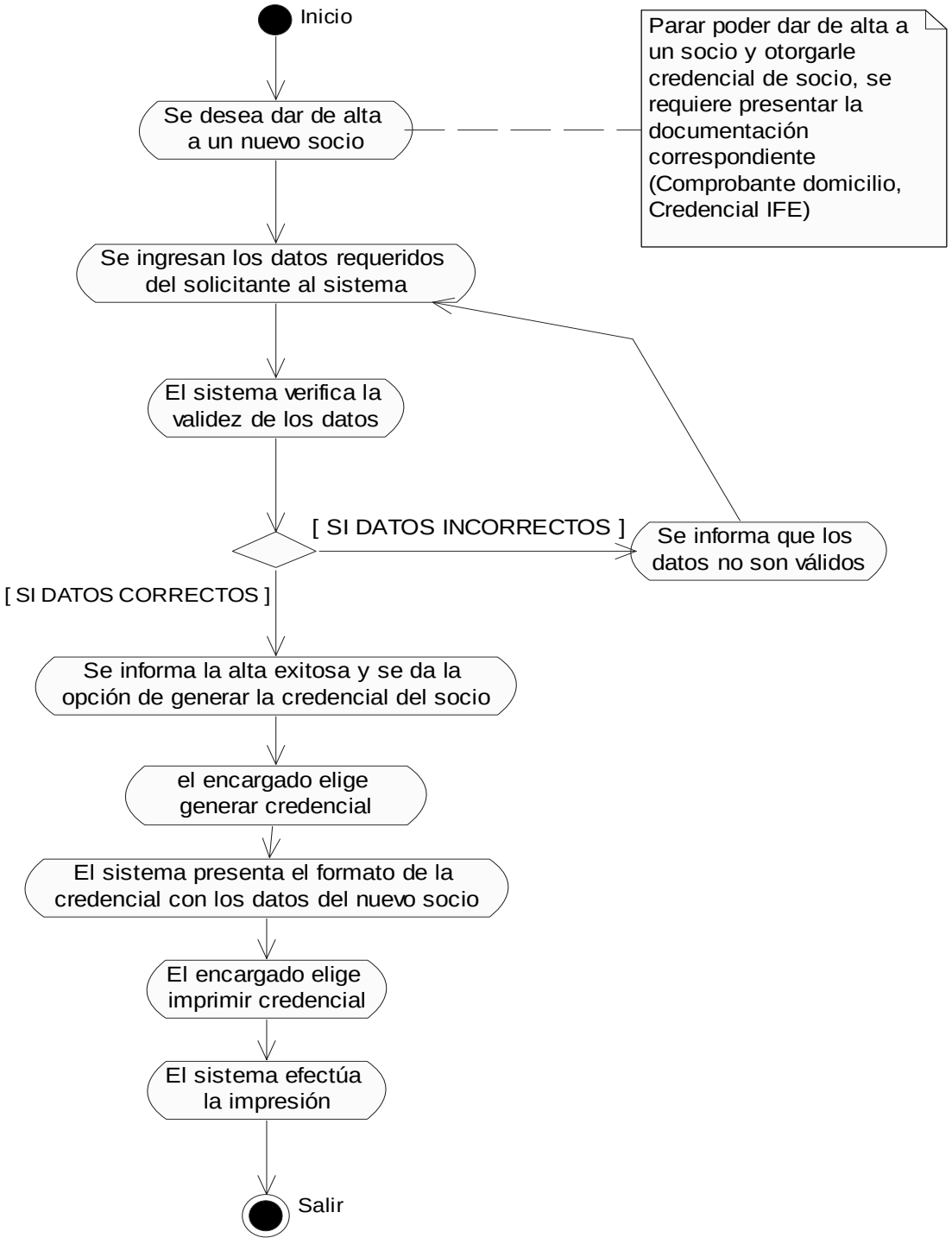


Figura 2.10. Secuencia de pasos efectuados para realizar una alta de socio.

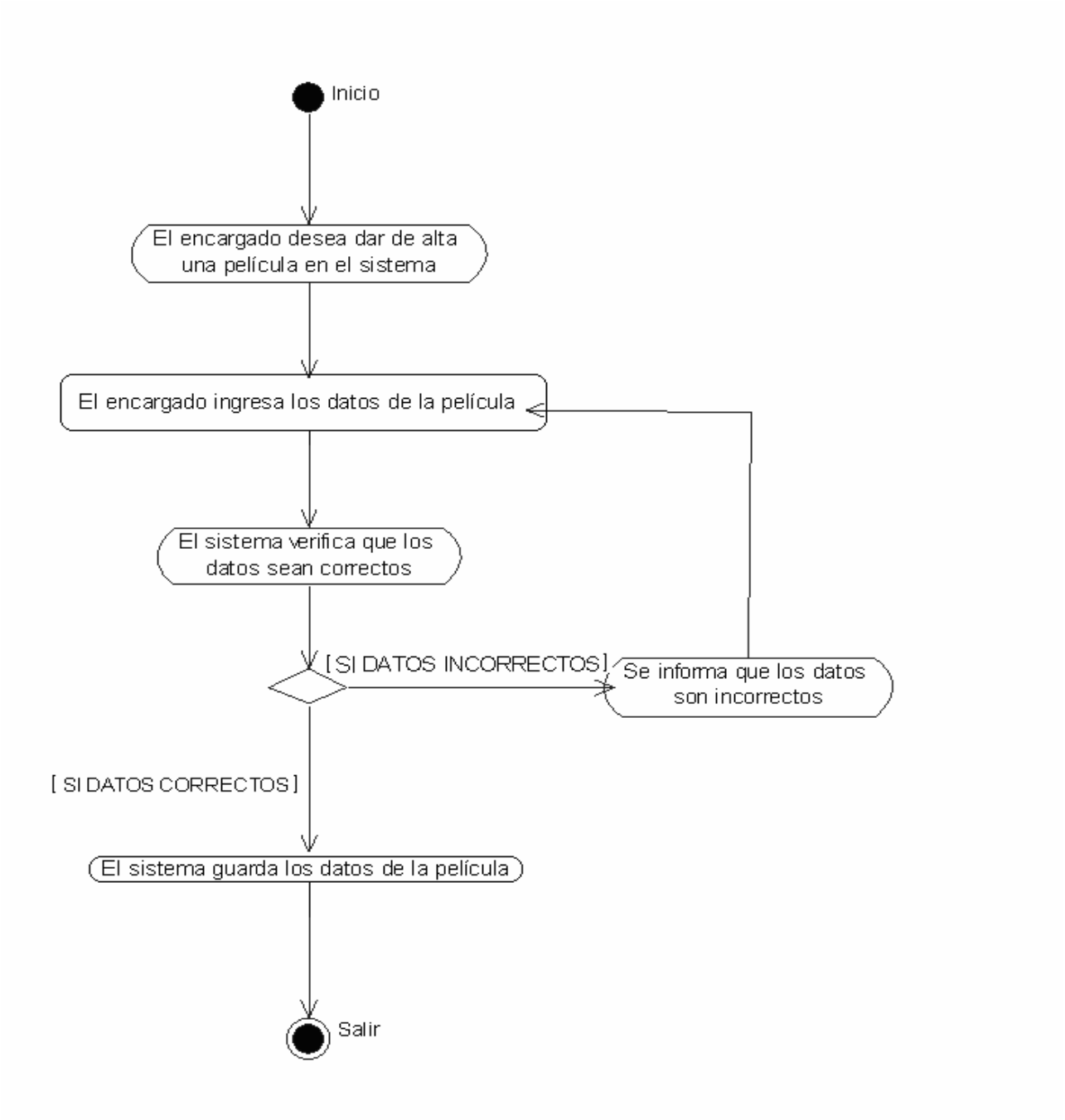


Figura 2.11. Secuencia de pasos efectuados para realizar una alta de película.

III. INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

PROGRAMACIÓN ORIENTADA A OBJETOS

Programación Tradicional

En la programación tradicional, conocida como *estructurada (imperativa)*, se separan los datos del programa de las funciones que los manipulan. El programa o aplicación completa, consiste de múltiples datos y múltiples funciones.

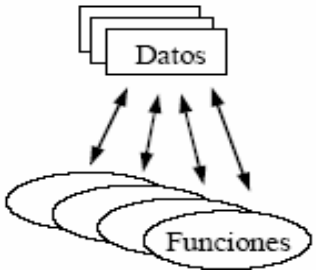


Figura 3.1. Esquema general de la programación estructurada.

Esta manera de programar tiene dos problemas principales. El primer problema es obligar a un programador a pensar como la máquina, en lugar de lo opuesto. El segundo problema es que toda la información presente es conocida y potencialmente utilizada por todas las funciones del programa y si se hiciera algún cambio en la estructura de alguno de los datos (se consideran todos como "globales"), potencialmente habría que modificar todas las funciones del programa para que éstas pudieran utilizar la nueva estructura.

Programación Orientada a Objetos

Al ser la unidad básica de programación el *objeto* y éste concepto del objeto nos acerca más a la manera de pensar de la gente al agregar un nivel de abstracción adicional donde internamente la estructura del programa se ajusta a la arquitectura de la máquina. En relación al segundo problema, los datos globales desaparecen, asignando a cada objeto sus propios datos y funciones locales, resultando en un programa o aplicación definido exclusivamente en término de objetos y sus relaciones entre sí.

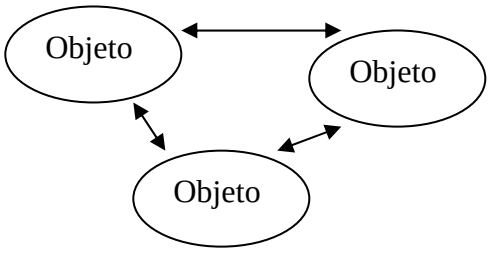


Figura 3.2. Objetos relacionados.

Obviamente debemos tener datos y funciones para que un programa tenga sentido, pero estos son guardados en cada objeto de manera independiente.

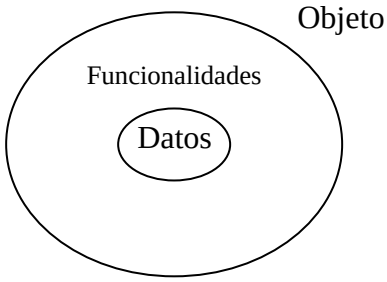


Figura 3.3. Organización de datos y funcionalidades.

Existen razones un poco más técnicas que motivan a la orientación a objetos, como son la *abstracción*, *modularidad*, *extensibilidad* y *reutilización*.

Abstracción. La idea básica de la abstracción es reducir el nivel de primitivas o representaciones básicas necesarias para producir un sistema de software. Con la programación orientada a objetos se definen dos niveles de abstracción. El nivel más alto, el de los objetos, es utilizado para escribir la aplicación mientras que el nivel más bajo, el de los datos y los métodos, es utilizado para describir sus detalles.
Por otro lado el objeto como estructura básica sirve para separar el "qué" de una aplicación del "cómo", o sea sus detalles, al contrario de la programación tradicional donde el "qué" y el "cómo" se resuelven a la vez.

Modularidad. La modularidad de un sistema depende de sus abstracciones básicas, lo cual permite dividir el sistema en componentes separados. Al tener abstracciones de mayor nivel la modularidad de los componentes también es de mayor nivel reduciendo el número final de componentes lo cual a su vez simplifica su manipulación y mantenimiento. Con la orientación a objetos, la modularidad del sistema se da en base a objetos.

Extensibilidad. La *extensibilidad* tiene como objetivo permitir cambios en el sistema de manera modular afectando lo mínimo posible el resto del sistema. Con la orientación a objetos, los cambios se dan a dos niveles: modificación externa e interna de los objetos. Los cambios internos a los objetos afectan principalmente al propio objeto, mientras que los cambios externos a los objetos afectarán de mayor forma al resto del sistema.

Reutilización. Una de las maneras de reducir la complejidad del software es mediante la *reutilización* o *reuso* de partes existentes. Mediante el reuso de código se puede aprovechar componentes genéricos para estructurar bibliotecas reutilizables, y así lograr una estandarización y simplificación de aplicaciones por medio de componentes genéricos prefabricados. Con la orientación a objetos, el *objeto* es la unidad de reuso más pequeña. A un nivel mucho mayor existen los *marcos de aplicación* ("frameworks") donde una aplicación genérica en un dominio particular se especializa para diferentes ambientes.

Características Mínimas de los Lenguajes Orientados a Objetos
Los cuatro conceptos básicos que hacen que un lenguaje sea considerado efectivamente orientado a objetos: *encapsulamiento*, *clasificación*, *generalización* y p*olimorfismo*.

Encapsulamiento. *E*s la separación de las propiedades externas de un objeto, o sea su interfase, correspondiente a la interfase de sus métodos, de los detalles de implementación internos del objeto, es decir sus datos y la implementación de sus métodos. Todo el detalle, al estar encapsulado, es desconocido por el resto de la aplicación, limitando el impacto de cualquier cambio en la implementación del objeto, ya que los cambios a las propiedades internas del objeto no afectan su interacción externa.

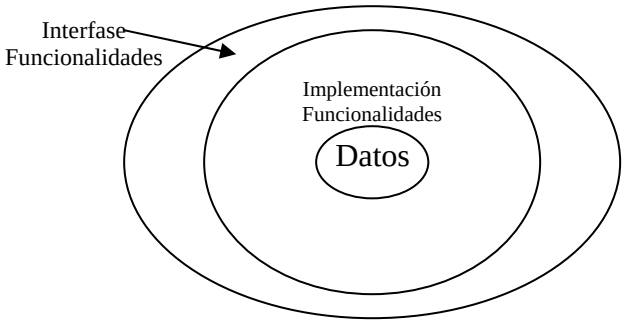


Figura 3.4. Representación del encapsulamiento.

Clasificación. Objetos que contienen estructuras similares, correspondiente a tipos de datos y métodos similares, se clasifican como pertenecientes a la misma *clase* de objeto. Nótese de que hablamos de *tipos* de datos similares, dado que los valores de los datos aún pueden cambiar en objetos de clase similar.

Generalización. Si tomamos como base la clasificación, y consideramos que no sólo los objetos similares pueden clasificarse, sino también las propias clases de los objetos, entonces se define la *generalización* o *especialización* de clases. Mediante la generalización, clases de objetos con estructura y comportamiento similar se reutilizan en la definición de las nuevas clases. Estas nuevas clases se consideran clases más especializadas o *subclases* mientras que las originales se consideran clases más generales o *superclases*. El mecanismo para describir jerarquías de generalización de clases se conoce como *herencia*, un término muy utilizado en la orientación a objetos, se dice que una subclase hereda de una superclase. La herencia es también una forma de reutilización de código, ya que se aprovechan descripciones de clases de objetos para luego definir clases de objetos parecidos.

Polimorfismo. Quizás el concepto más complicado de explicar y en cierta manera el más poderoso es el *polimorfismo*. De manera simplificada, mediante el polimorfismo se definen métodos con el mismo nombre e interfaz en distintas clases de objetos, pero bajo implementaciones distintas. ¿Para qué sirve entonces el polimorfismo?, el polimorfismo es útil para extender la funcionalidad existente en los objetos del sistema, a nuevos objetos aún desconocidos en ese momento. Es como definir un estándar de interfaces para los objetos la cual debe ser seguida por todos los existentes y nuevos.

Haciendo una analogía, todo el tiempo aparecen nuevas aplicaciones de software que pueden ejecutarse en sistemas ya existentes. Para que todo funcione correctamente el diseñador del nuevo software debe mantener un estándar en las interfaces de sus

métodos que sea ya conocida y aceptada aunque la implementación de los métodos sea obviamente distinta.

INTRODUCCIÓN A JAVA

Java es un lenguaje originalmente desarrollado por un grupo de ingenieros de Sun MicroSystems, utilizado por el explorador Netscape posteriormente como base para Javascript. Si bien su uso se destaca en el Web, sirve para crear todo tipo de aplicaciones (locales, intranet o internet).

CARACTERÍSTICAS

- Gestiona la memoria automáticamente
- Multithreading (permite muchas actividades simultáneas en un programa).
- Mecanismos de seguridad incorporados
- Herramientas de documentación incorporadas

Orientado a Objetos – Es un lenguaje orientado a objetos, por lo tanto, se cuenta con un ligado dinámico de clases en tiempo de ejecución, herencia y polimorfismo.

Portátil – Uno de los aspectos que han hecho de Java un lenguaje muy utilizado es su portabilidad. Java es exactamente igual bajo cualquier plataforma. La gran importancia de este aspecto es que si se compila el programa bajo una plataforma particular, el sistema correrá en cualquier máquina, reduciendo mucho el costo de desarrollo (tiempo y dinero); esto gracias a su arquitectura de máquina virtual de java (JVM).

Abierto – El aspecto de portabilidad se da gracias a su diseño abierto que permite a cualquier compañía, e incluso desarrollador, tomar el código fuente, para adaptarlo a una nueva plataforma donde aún no se ha probado.

Integrado al Web – Esta ha sido la razón para su gran difusión en una época donde el Internet ha sido de tanta importancia.

Simple – Un programa en Java no contiene más que clases, simplificando el programa y al propio compilador. Java elimina la aritmética de apuntadores lo cual agrega mucha complejidad en la administración de memoria. También se eliminan aspectos de manejo complicado como es la herencia múltiple. Además de su número de palabras clave reducido.

Robusto –Java también incluye manejo de excepciones y recolección de basura para lograr programas más robustos.

Seguro – La seguridad es apoyada por el modelo de verificación de código en tiempo de ejecución.

Eficiencia –Esta eficiencia se basa, en que se cuenta con un compilador para la generación de código.

Bibliotecas –Java contiene un sinfín de bibliotecas que facilitan en gran manera la creación de programas, además de asegurar una estandarización entre aplicaciones.

Tecnología – Aparte de Java como lenguaje se cuenta con productos tales como *EJB* (Enterprise JavaBeans), *JSP* (Java Server Pages), Java Servlets y *JDBC* (Java Data Base Connectors). Además existen productos relacionados con estándares tales como *CORBA* (Common Object Request Brower Architecture) y *XML* (eXtended Markup Language).

Compilación
Se escribe un programa en código Java. Este programa, que tiene como extensión el sufijo ".java", es compilado por cualquiera de los compiladores de Java en alguna de las distintas plataformas. En general debe existir un archivo ".java" por cada clase que exista en el programa, donde el archivo debe tener el mismo nombre que la clase contenida. El compilador genera el código final, conocido como *bytecode*, a ser interpretado por la máquina virtual de Java. El programa generado tiene como extensión el sufijo ".class". Se genera un archivo ".class" por cada clase que se tenga en la aplicación.

Ejecución
Durante la ejecución se obtiene el *bytecode*, guardado en los archivos ".class", que puede ya estar en la plataforma actual o haber sido enviado por la red, como en el caso de un *browser*. El *bytecode* es cargado en la máquina virtual por el cargador de clases. A continuación este código es verificado por el verificador de *bytecode*, y dependiendo del hardware con que se cuenta, puede ser interpretado y ejecutado por el procesador virtual de la máquina virtual o traducido a código de un procesador de Java mediante el generador de código.

Paquetes
Java lleva a un nuevo nivel el concepto de bibliotecas o *paquetes*. Estos paquetes proveen una amplia funcionalidad para crear nuevas aplicaciones para Java. Además de servir como bibliotecas, definen un *API* (Application Program Interface) que permite al desarrollador extender las clases de estos paquetes para adaptarlos a las necesidades básicas de un programa. Java organiza estos paquetes en componentes jerárquicos a partir de dos directorios raíz principales.
Los paquetes del API de Java se dividen en básicos y opcionales. Los paquetes del API de Java comienzan ya sea con "java"(paquetes básicos) o javax (paquetes opcionales).

EL ENTORNO DE DESARROLLO DE JAVA

Existen distintos programas comerciales que permiten desarrollar código *Java*. La compañía *Sun Microsystems*, creadora de *Java*, distribuye gratuitamente el *Java(tm) Development Kit* (*JDK*). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en *Java*.

Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (es el denominado *Debugger*). Existe también una versión reducida del *JDK*, denominada *JRE* (*Java Runtime Environment*) destinada únicamente a ejecutar código *Java* (no permite compilar).

El compilador de Java

Se trata de una de las herramientas de desarrollo incluidas en el *JDK*. Realiza un análisis de sintaxis del código escrito en los ficheros fuente de *Java* (con extensión **.java*). Si no encuentra errores en el código genera los ficheros compilados (con extensión **.class*). En otro caso muestra la línea o líneas erróneas. En el *JDK* de *Sun* dicho compilador se llama *javac.exe*.

La Máquina Virtual de Java (Java Virtual Machina)

La existencia de distintos tipos de procesadores y computadoras llevó a los ingenieros de *Sun Microsystems* a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se plantea la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "*máquina hipotética o virtual*", denominada *Java Virtual Machina* (*JVM*). Es esta *JVM* quien *interpreta* este código neutro convirtiéndolo a código particular de la CPU o chip utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.

La *JVM* es el intérprete de *Java*. Ejecuta los "*bytecodes*" (ficheros compilados con extensión **.class*) creados por el compilador de *Java* (*javac.exe*). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado *JIT* (*Just-In- Time Compiler*), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

Los ***IDEs* (*Integrated Development Environment*),** son entornos de desarrollo integrados. En un mismo programa es posible escribir el código *Java*, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar detección y *corrección de errores* gráficamente, frente a la versión que incorpora el *JDK* basada en la utilización de una consola bastante difícil y pesada de utilizar.
Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con *componentes* ya desarrollados, los cuales se incorporan al proyecto o programa. Como inconvenientes se pueden señalar algunos fallos de compatibilidad entre plataformas y ficheros resultantes de mayor tamaño que los basados en clases estándar. Algunos de los IDEs más conocidos son Netbeans, Eclipse, JDeveloper 10g, etc.

SINTAXIS DEL LENGUAJE JAVA

Empezaremos con un pequeño y simple ejemplo de un programa en java que imprime un texto.

```
// Programa que imprime texto.
public class Hola {
// el método main empieza la ejecución de la aplicación de java
        public static void main( String args[] )
        {
                System.out.println( "Programando en Java" );
        } // fin del método main
} // fin de la clase Hola
```

Operadores aritméticos

Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: *suma* (+), *resta* (-), *multiplicación* (*), *división* (/) y *resto de la división* (%).

Operadores de asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación por excelencia es el *operador igual* (=). La forma general de las sentencias de asignación con este operador es: variable = expresión;
Java dispone de otros operadores de asignación. Se trata de versiones abreviadas del operador (=) que realizan operaciones “acumulativas” o de “decremento” sobre una variable (+=, -=).

Operadores relacionales

Los *operadores relacionales* sirven para realizar comparaciones de igualdad (= =), desigualdad (!=) y relación de menor (<) o mayor (>), mayor o igual (>=), menor o igual (<=). El resultado de estos operadores es siempre un valor *boolean* (*true* o *false*) según se cumpla o no la relación considerada.

CLASES Y OBJETOS EN JAVA

Un objeto, puede verse como una pieza de software que cumple con ciertas características:

- Encapsulamiento
- Abstracción

Encapsulamiento significa que el objeto es auto-contenido, o sea que la misma definición del objeto incluye tanto los datos que éste usa (atributos) como los procedimientos (métodos) que actúan sobre los mismos.
Cuando se utiliza programación orientada a objetos, se definen clases (que definen objetos genéricos) y la forma en que los objetos interactúan entre ellos, a través de mensajes. Al crear un objeto de una clase dada, se dice que se crea una instancia de la clase, o un objeto propiamente dicho. Por ejemplo, una clase podría ser "autos", y un auto dado es una instancia de la clase.

En cuanto a la **abstracción**, simplemente significa que se reduce el nivel de representaciones básicas necesarias.

Declaración de la clase
La clase se declara mediante la línea public class NombreClase. En el caso más general, la declaración de una clase puede contener los siguientes elementos:

[public] [final | abstract] class Clase [extends ClaseMadre] [implements Interfase1 [, Interfase2]…]

o bien, para interfaces:

[public] interface Interfase [extends InterfaseMadre1 [, InterfaseMadre2]…]

Como se ve, lo único obligatorio es class y el nombre de la clase.

Public, final o abstract
Definir una clase como pública (public) significa que puede ser usada por cualquier clase en cualquier paquete. Si no lo es, solamente puede ser utilizada por clases del mismo paquete.
Una clase final (final) es aquella que no puede tener clases que la hereden.
Una clase abstracta (abstract) es una clase que puede tener herederas, pero no puede ser instanciada. Es, literalmente, abstracta (como la clase Number definida en java.lang).
¿Para qué sirve? Para modelar conceptos.

Extends
La instrucción extends indica de qué clase desciende la nuestra. Si se omite, java asume que desciende de la superclase Object.
Cuando una clase desciende de otra, esto significa que hereda sus atributos y sus métodos (es decir que, a menos que los redefinamos, sus métodos son los mismos que los de la clase madre.

Implements
Una interfase (interface) es una clase que declara sus métodos pero no los implementa; cuando una clase implementa (implements) una o más interfases, debe contener la implementación de todos los métodos (con las mismas listas de parámetros) de dichas interfases.

Interface
Una interfase (interface), es una clase que no implementa sus métodos sino que deja a cargo la implementación a otras clases. Las interfases pueden, asimismo, descender de otras interfases pero no de otras clases.
Todos sus métodos son por definición abstractos y sus atributos son finales (aunque esto no se indica en el cuerpo de la interfase).
Son útiles para generar relaciones entre clases que de otro modo no están relacionadas.

El cuerpo de la clase

El cuerpo de la clase, encerrado entre { y }, es la lista de *atributos* (variables) y *métodos* (funciones) que constituyen la clase.
No es obligatorio, pero en general se listan primero los atributos y luego los métodos.

Declaración de atributos

En java no hay variables globales; todas las variables se declaran dentro del cuerpo de la clase o dentro de un método. Las variables declaradas dentro de un método son locales al método; las variables declaradas en el cuerpo de la clase se dice que son miembros de la clase y son accesibles por todos los métodos de la clase.

Finalmente, los atributos miembros de la clase pueden ser atributos de clase o atributos de instancia; se dice que son atributos de clase si se usa la palabra clave static: en ese caso la variable es única para todas las instancias (objetos) de la clase (ocupa un único lugar en memoria). Si no se usa static, el sistema crea un lugar nuevo para esa variable con cada instancia (o sea que es independiente para cada objeto).

La declaración sigue siempre el mismo esquema:

[private|protected|public] [static] [final] Tipo NombreVariable [= Valor];

Private, protected o public
Java tiene cuatro tipos de acceso diferente a las variables o métodos de una clase: privado, protegido, público, friendly o por paquete (si no se especifica nada).
De acuerdo a la forma en que se especifica un atributo, los objetos de otras clases tienen distintas posibilidades de accederlos:

Static y final
Static sirve para definir un atributo como de clase, o sea único para todos los objetos de la clase.
En cuanto a *final*, como en las clases, determina que un atributo no pueda ser sobreescrito o redefinido.

De acuerdo a ésto java clasifica los datos de la siguiente manera:

TIPOS DE DATOS	▪ Primitivos	- Numéricos - Alfanuméricos
	▪ Referencia	- Clases - Arrays

Los tipos de datos primitivos son:

	Tipo	Descripción	Tamaño/Formato
NUMÉRICOS	byte	Entero de longitud byte	8-bits
	short	Entero corto	16 bits
	int	Entero	32 bits
	long	Entero largo	64bits
	float	Punto flotante de precisión simple	32-bit IEEE 754
	double	Punto flotante de precisión doble	64-bit IEEE 754
	boolean	Valor booleano	True/False
ALFANUMÉRICOS	char	Un carácter	16-bit carácter unicode

Los arrays son arreglos de cualquier tipo (básico o no).
Los arreglos siempre son dinámicos, por lo que no es válido poner algo como:
Integer cadena[5];

Aunque sí es válido inicializar un arreglo, como en:
int días[ ] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
char letras[ ] = { 'E', 'F', 'M', 'A', 'M', 'J', 'J', 'A', 'S', 'O', 'N', 'D' };
String nombres [ ] = new String[12];

En Java, para todas las variables de tipo primitivo se accede al valor asignado a la misma directamente (no se conoce la dirección de memoria que ocupa). Para las demás (arrays, clases o interfases), se accede a través de un puntero a la variable. El valor del puntero no es accesible ni se puede modificar.

Declaración de métodos

Los métodos, como las clases, tienen una declaración y un cuerpo.
La declaración es del tipo:

```
[private|protected|public] [static] [abstract] [final] [native] [synchronized]
TipoDevuelto NombreMétodo ( [tipo1 nombre1[, tipo2 nombre2 ]…] ) [throws
excepción1 [,excepción2]… ]
```

Los métodos implementan, a través de funciones, operaciones y estructuras de control, el cálculo de algún parámetro, que es el que devuelven al objeto que los llama. Sólo pueden devolver un valor (del tipo TipoDevuelto), aunque pueden no devolver ninguno (en ese caso TipoDevuelto es void).
Los métodos pueden utilizar valores que les pasa el objeto que los llama (parámetros), indicados con tipo1 nombre1, tipo2 nombre2… en el esquema de la declaración.

Veamos un pequeño ejemplo:

```
public int AumentarCuenta(int cantidad) {
        cnt = cnt + cantidad;
        return cnt;
}
```

En detalle:
- El método recibe un valor entero (cantidad)
- Lo suma a la variable de instancia cnt
- Devuelve la suma (return cnt)

Public, private y protected actúan exactamente igual para los métodos que para los atributos.
Los métodos estáticos (static), son, como los atributos, métodos de clase; si el método no es static es un método de instancia. El significado es el mismo que para los atributos: un método static es compartido por todas las instancias de la clase.
Los métodos abstractos (abstract) son aquellos de los que se da la declaración pero no la implementación (o sea que consiste sólo del encabezado). Cualquier clase que contenga al menos un método abstracto (o cuya clase madre contenga al menos un método abstracto que no esté implementado en la hija) es una clase abstracta.
Es final un método que no puede ser redefinido por ningún descendiente de la clase.

Finalmente, la cláusula throws sirve para indicar que la clase genera determinadas excepciones.

El cuerpo de los métodos
Dentro de los métodos pueden incluirse:
- Declaración de variables locales
- Asignaciones a variables
- Operaciones matemáticas
- Llamados a otros métodos:
  - dentro de la clase
  - de instancia, de otras clases
  - de clase, de cualquier clase
- Estructuras de control
- Excepciones (try, catch, que veremos más adelante)

Declaración de variables locales
Las variables locales se declaran igual que los atributos de la clase:

```
Tipo NombreVariable [= Valor];
Ej:     int suma;
        float precio;
        Contador laCuenta;
```

Sólo que aquí no se declaran private, public, etc., sino que las variables definidas dentro del método sólo son accesibles por él.

Las variables pueden inicializarse al crearse:

```
Ej:     int suma = 0;
        float precio = 12.3;
        Contador laCuenta = new Contador ( );
```

Asignaciones a variables

Se asigna un valor a una variable mediante el signo =:

```
Variable = Constante | Expresió ;
Ej:     suma = suma + 1;
        precio = 1.05 * precio;
```

ESTRUCTURAS DE CONTROL

If-else

La más común de todas, permite ejecutar una instrucción (o secuencia de instrucciones) si se da una condición dada (o, mediante la cláusula else, ejecutar otra secuencia en caso contrario).

```
if (expresión_booleana) instrucción_si_true;
[else instrucción_si_false;]
o bien:
        if (expresión_booleana) {
                instrucciones_si_true;
        }
        else {
                instrucciones_si_false;
        }
```

Switch…case…brake…default

Permite ejecutar una serie de operaciones para el caso de que una variable tenga un valor entero dado. La ejecución saltea todos los case hasta que encuentra uno con el valor de la variable, y ejecuta desde allí hasta el final del case o hasta que encuentre un break, en cuyo caso salta al final del case. El default permite poner una serie de instrucciones que se ejecutan en caso de que la igualdad no se de para ninguno de los case.

```
        switch (expresión_entera) {
                case (valor1): instrucciones_1;
                                [break;]
                case (valor2): instrucciones_2;
                                [break;]
                …..
                case (valorN): instrucciones_N;
                                [break;]
                default: instrucciones_por_defecto;
        }
```

While

Permite ejecutar un grupo de instrucciones repetidamente mientras se cumpla una condición dada:

```
while (expresión_booleana) {
        instrucciones…
}
```

Do…while

Similar al anterior, sólo que la condición se evalúa al final del ciclo y no al principio:

```
do {
        instrucciones…
} while (expresión_booleana);
```

For

También para ejecutar en forma repetida una serie de instrucciones; es un poco más complejo:

```
for ( instrucciones_iniciales; condición_booleana; instruccion_repetitiva_x ) {
        instrucciones…
}
```

Si bien las instrucciones pueden ser cualquiera (el bucle se repite mientras la condición sea verdadera), lo usual es utilizarlo para "contar" la cantidad de veces que se repiten las instrucciones; se podría indicar así:

```
for ( contador = valor_inicial; contador < valor_final; contador++ ) {
        instrucciones…
}
```

Break y continue

Estas instrucciones permiten saltar al final de una ejecución repetitiva (break) o al principio de la misma (continue).

COMENTARIOS

En java hay tres tipos de comentarios:

//Comentarios para una sola línea

/*Comentarios de una o más líneas
*/

/**Comentarios de documentación, de una o más líneas
*/

Los comentarios de documentación, colocados inmediatamente antes de una declaración (de variable o función), indican que ese comentario ha de ser colocado en la documentación que se genera automáticamente cuando se utiliza la herramienta de javadoc. Dichos comentarios sirven como descripción del elemento declarado permitiendo generar una documentación de nuestras clases al mismo tiempo que se genera el código.

En este tipo de comentario para documentación, se permite la introducción de algunos tokens o palabras clave, que harán que la información que les sigue aparezca en forma diferente al resto en la documentación.

EXCEPCIONES

Las excepciones en java están destinadas para la detección y corrección de errores. Si hay un error la aplicación no debería morirse o generar un crash. Se debería lanzar una excepción que nosotros deberíamos capturar y resolver la situación de error.
Utilizadas adecuadamente las excepciones aumentan la robustez de las aplicaciones.
Para que un método en java pueda lanzar excepciones hay que indicarlo expresamente.

```
Void metodoFallo() throws NullPointerException, CaidaException
```

Se pueden definir excepciones propias, bastará con extender la clase Exception y proporcionar la funcionalidad extra.

Las excepciones lanzadas por un método deben recogerse en un bloque try/catch o try/finally.
La estructura queda así:

try *{*

Bloque de código donde se prevé que se genere una excepción.

*}****catch****{*

Es el código que se ejecuta cuando se produce la excepción.

*}****finally****{*

Es el bloque de código que se ejecuta siempre, haya o no excepción.

}

IV. DISEÑO DE APLICACIONES CON EL MODELO DE TRES CAPAS

Definición del modelo de tres capas

Generalmente, no es necesario inventar una nueva arquitectura software para cada sistema de información. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto.

Así, las arquitecturas más universales son:

- Monolítica. Donde el software se estructura en grupos funcionales muy acoplados.
- Cliente-servidor. Donde el software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones.
- Arquitectura de tres niveles. Especialización de la arquitectura cliente-servidor donde la carga se divide en tres partes con un reparto claro de funciones: una capa para la presentación, otra para el cálculo y otra para el almacenamiento. Una capa solamente tiene relación con la siguiente.

En los últimos tiempos, para el desarrollo de aplicaciones Web, se viene imponiendo el empleo de tecnologías de componentes donde es habitual realizar una división de la arquitectura de las aplicaciones en tres capas funcionales:

- Capa de Presentación
- Capa de Datos
- Capa de Negocio

La **capa de presentación** es lo que tradicionalmente se conoce como Interfaz de Usuario. Son las ventanas o páginas Web, por decirlo de alguna manera. Presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio.

En la **capa de negocios** es donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

Sin embargo, el acceso a los repositorios de información no se realiza directamente, sino que se hace a través de la **capa de datos**, encargada de proporcionar dicho servicio.

La **capa de datos** es donde residen los datos. Está formada por uno o más gestor de bases de datos que realiza todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

El diseño de una aplicación de acuerdo con esta arquitectura de 3 capas presenta la **gran ventaja** de producir código modular en el que la modificación de uno de sus componentes, por ejemplo la interfaz de usuario (la forma en la que el usuario navega por el Web o la forma en la que se presentan los datos) no requiere modificar ninguno de los elementos de las otras capas. Asimismo, si por cualquier razón se migrase el repositorio de información y se alojase en una base de datos diferente, únicamente sería necesario modificar la capa de datos.

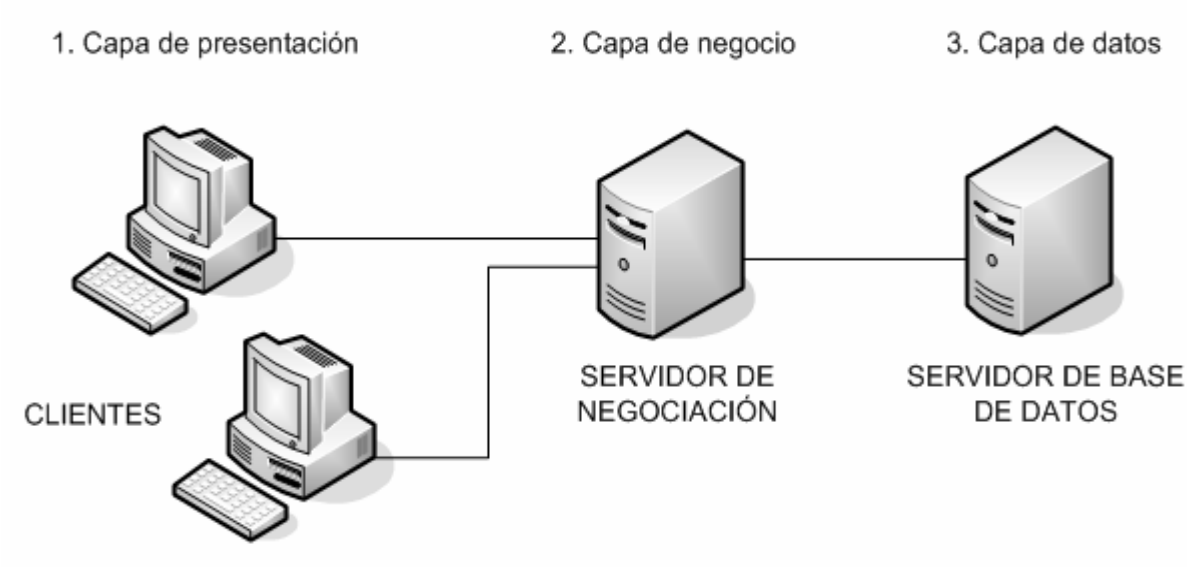


Figura 4.1. Esquema del Modelo de Tres Capas.

Además permite distribuir el trabajo de creación de una aplicación por niveles, de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, simplemente es necesario conocer la API que existe entre niveles.

El diseño de aplicaciones Web siguiendo la arquitectura funcional antes descrita se refleja posteriormente en el modelo lógico. La capa de presentación se implementa según el patrón de diseño Modelo-Vista-Controlador (MVC), cuyo principal objetivo es separar la presentación de la lógica de negocio.

Capa de Presentación

Los servicios de presentación proporcionan la interfaz necesaria para presentar información y reunir datos. También aseguran los servicios de negocios necesarios para ofrecer las capacidades de transacciones requeridas e integrar al usuario con la aplicación para ejecutar un proceso de negocios.

Los servicios de presentación generalmente son identificados con la interfaz de usuario, y normalmente residen en un programa ejecutable localizado en la estación de trabajo del usuario final. Aún así, existen oportunidades para identificar servicios que residen en componentes separados.

El cliente proporciona el contexto de presentación, generalmente un *explorador* como Microsoft Internet Explorer o Netscape Navigator, que permite ver los datos remotos a través de una capa de presentación HTML.

La capa de servicios de presentación es responsable de:

- Obtener información del usuario.
- Enviar la información del usuario a los servicios de negocios para su procesamiento.
- Recibir los resultados del procesamiento de los servicios de negocios.
- Presentar estos resultados al usuario.

INTERFAZ

La interfaz es el medio por el cual el usuario puede hacer todas sus operaciones, por lo que su diseño supone:

- Idear, imaginar la página, definir cómo será.
- Identificar los elementos que la constituirán: imágenes, colores, referencias, audio, ligas, las partes de la página.
- Definir el aspecto visual.
- Definir el contenido informativo
- Definir los recursos de interacción del usuario con el sistema u otros usuarios.
- Definir nexos o ligas.

Proceso de diseño de la interfaz

El diseño de la interfaz se basa en los procesos y tareas de los usuarios, de lo contrario el sistema sería un fracaso; pero a pesar de que la interfaz va a estar basada en el Web, esto no quiere decir que deba tener todos los aspectos llamativos de las páginas comunes, se debe sensibilizar para ofrecer únicamente lo necesario para que el usuario trabaje. Para esto se deben seguir los siguientes pasos:

a. Entender quién usará el sistema y para qué. Esta información se obtiene de la definición de casos de uso y perfiles de usuario.
b. Utilizar un estándar o normatividad de la empresa. Aunque exista un estándar a seguir, puede ignorarse si interfiere con las formas o los procesos.
c. Bosquejar un diseño. Sirve para darnos una idea de cómo se verá la interfaz, puesto que forza a pensar en los elementos de las pantallas y su orden.

d. Crear un prototipo. El cual se presente a los usuarios sin conexiones a bases de datos, para verificar la congruencia de la interfaz.
e. Evaluarla con los usuarios. Existen muchos problemas que sólo aparecerán cuando el diseño se está probando con los usuarios.
f. Construcción. Una vez comprobada la congruencia del diseño se puede empezar a construir la interfaz.

Criterios de usabilidad

- Diseño amigable
- Congruencia con los procesos
- Distribución lógica
- Navegación lógica
- Retroalimentación
- Consistencia
- Implementación de ayuda

HTML

Un documento escrito en HTML contiene básicamente los siguientes elementos:

<HTML>	Indica el inicio del documento.
<HEAD>	Inicio de la cabecera.
<TITLE>	Inicio del título del documento.
</TITLE>	Final del título del documento.
</HEAD>	Final de la cabecera del documento.
<BODY>	Inicio del cuerpo del documento.
</BODY>	Final del cuerpo del documento.
</HTML>	Final del documento.

Sintaxis del HTML

HTML basa su sintaxis en un elemento de base al que llamamos etiqueta o tag. La etiqueta presenta frecuentemente dos partes:

Una apertura de forma general <etiqueta>
Un cierre de tipo </ etiqueta>

Todo lo incluido en el interior de esa etiqueta sufrirá las modificaciones que caracterizan a esta etiqueta. Así por ejemplo:

Las etiquetas y definen un texto en negrita. Si en nuestro documento HTML escribimos una frase con el siguiente código:

Esto esta en negrita
El resultado Será: **Esto esta en negrita**

```
<html>

<head>
<title>Cocina Para Todos</title>
</head>
<body>
<p><b>Bienvenido,</b></p>
<p>Aquí hay recetas deliciosas.</p>
```

```
</body>
</html>
```

Listas

Las listas sirven para enumerar y definir elementos.
Podemos distinguir tres tipos de listas:

- Listas desordenadas. Son delimitadas por las etiquetas <ul> y . Cada uno de los elementos de la lista es citado por medio de una etiqueta <li>.

Países del mundo Argentina México 	▪ Argentina ▪ México

- Listas ordenadas. En este caso usaremos las etiquetas <ol> y su cierre. Cada elemento será igualmente precedido de su etiqueta <li>. (Aparecen enumeradas).
- Listas de definición. Cada elemento es presentado junto con su definición. La etiqueta principal es <dl> y </dl>. La etiquetas del elemento y su definición son <dt> y <dd> respectivamente. (Aparece con sangría).

Por otro lado la sintaxis general de un enlace es de la forma: *contenido*
La sintaxis de una imagen: <img src="camino hacia el archivo">

Tablas en HTML

Una tabla es un conjunto de celdas organizadas dentro de las cuales podemos alojar distintos contenidos.

Las tablas son definidas por las etiquetas <table> y </table>.

Las tablas son descritas por líneas de izquierda a derecha. Cada una de estas líneas es definida por otra etiqueta y su cierre: <tr> y </tr>.

Asimismo, dentro de cada línea, habrá diferentes celdas. Cada una de estas celdas será definida por otro par de etiquetas: <td> y </td>. Dentro de estas etiquetas será donde coloquemos nuestro contenido.

```
<table>
   <tr>
      <td>Celda 1, linea 1</td>
      <td> Celda 2, linea 1</td>
   </tr>
</table>
```

El resultado:

Celda 1, linea 1	Celda 2, linea 1

Formularios HTML

Con ellos podemos intercambiar información con quien interactúe con la página. Para comprar un artículo, rellenar una encuesta...

Los formularios son esas famosas cajas de texto y botones que podemos encontrar en muchas páginas Web. Los datos que el usuario introduce en estos campos son enviados a un programa que se encarga de procesarlo automáticamente. Los formularios son definidos por medio de las etiquetas <form> y </form>. Entre estas dos etiquetas colocaremos todos los campos y botones que componen el formulario. Dentro de esta etiqueta <form> debemos especificar algunos atributos:

action
Define el tipo de acción a llevar a cabo con el formulario. El formulario es enviado a un programa o script que procesa su contenido <form action="dirección del archivo" ...>

method
Este atributo se encarga de especificar la forma en la que el formulario es enviado. Los dos valores posibles que puede tomar este atributo son post y get.

Elementos de Formularios

Caja de texto
Las cajas de texto son colocadas por medio de la etiqueta <input>. Dentro de esta etiqueta hemos de especificar el valor de dos atributos: type y name. La etiqueta es de la siguiente forma: <input type="text" name="nombre">

Password
Podemos esconder el texto escrito por medio de asteriscos u otro símbolo de manera a aportar una cierta confidencialidad: <input type="password" name="nombre">

Textarea
Es un espacio de escritura compuesto de varias líneas, para cometarios u opiniones, se usa la etiqueta: <textarea> y su cierre correspondiente. <textarea name="comenta"></textarea>

Menús desplegables
Nos permiten elegir una (o varias) de las múltiples opciones que nos proponen. Para construirlas emplearemos una etiqueta con su respectivo cierre: <select> Cada opción será incluida en una línea precedida de la etiqueta <option>.

```
<select name="estacion">
   <option>Invierno</option>
   <option>Verano</option>
</select>
```

Botones de radio
Permite elegir únicamente una de las opciones que se le proponen. La etiqueta empleada en este caso es <input> en la cual tendremos el atributo type ha de tomar el valor radio.

```
<input type="radio" name="estacion" value="1">Invierno
<br>
<input type="radio" name="estacion" value="2">Verano
```

Checkbox
Pueden ser activados o desactivados por el usuario con un simple clic sobre la caja en cuestión. La sintaxis es <input type="checkbox"name="p">Programa

Submit

Para dar por finalizado el proceso de relleno del formulario y enviarlo, se ha de validar por medio de este tipo de botón previsto a tal efecto. <input type="submit" value="Enviar">

Reset

Este botón nos permite borrar el formulario por completo en el caso de desear rehacerlo desde el principio. <input type="reset" value="Borrar">

Hidden

Ayudan al programa en su procesamiento del formulario. Este tipo de datos no se muestran en la página pero si pueden ser detectados solicitando el código fuente. <input type=hidden name="sitio" value="www.oculto.com">

Button

Son pulsables como cualquier otro botón. El uso más frecuente de un botón es en la programación en el cliente. Utilizando lenguajes como Javascript podemos definir acciones a tomar cuando se pulse el botón de una página Web. <input type=button value="Texto escrito en el botón">

HOJAS DE ESTILO (CSS)

El lenguaje HTML está limitado a la hora de aplicarle formato a un documento. Esto es así porque fue concebido para otros usos (científicos sobretodo), distinto a los actuales, mucho más amplios.

El modo de funcionamiento de las CSS consiste en definir, mediante una sintaxis especial, la forma de presentación que le aplicaremos a:

- Un Web entero, de modo que se puede definir la forma de todo el Web de una sola vez.
- Un documento HTML o página, se puede definir la forma, en un pequeño trozo de código en la cabecera, a toda la página.
- Una porción del documento, aplicando estilos visibles en un trozo de la página.
- Una etiqueta en concreto, llegando incluso a poder definir varios estilos diferentes para una sola etiqueta. Esto es muy importante ya que ofrece potencia en nuestra programación. Podemos definir, por ejemplo, varios tipos de párrafos: en rojo, en azul, con márgenes, sin ellos...

Pequeñas partes de la página

Para definir estilos en secciones reducidas de una página se utiliza la etiqueta <SPAN>. Con su atributo style indicamos en sintaxis CSS las características de estilos. Lo vemos con un ejemplo, pondremos un párrafo en el que determinadas palabras las vamos a visualizar en color verde.

```
Esto es un párrafo con palabras <SPAN style="color:green">en color verde</SPAN>.
```

Estilo definido para una etiqueta

De este modo podemos hacer que toda una etiqueta muestre un estilo determinado. Utilizamos el atributo style, que es admitido por todas las etiquetas del HTML.

```
<p style="color:#990000">
Esto es un párrafo de color rojo.</p>
<p style="color:#000099">
Esto es un párrafo de color azul.</p>
```

Estilo definido en una parte de la página

Con la etiqueta <DIV> podemos definir secciones de una página y aplicarle estilos con el atributo style, es decir, podemos definir estilos de una vez a todo un bloque de la página.

```
<div style="color:#000099; font-weight:bold">
<h3>Estas etiquetas van en <i>azul y negrita</i></h3>
<p>Seguimos dentro del DIV, luego permanecen los etilos </p>
</div>
```

Estilo definido para toda una página

Podemos definir, en la cabecera del documento, estilos para que sean aplicados a toda la página.

A grandes rasgos, entre de <STYLE> y </STYLE>, se coloca el nombre de la etiqueta a la que queremos definir los estilos y entre llaves -{ }- colocamos en sintaxis CSS las características de estilos.

```
<html>
<head>
  <STYLE type="text/css">
 <!--
 H1 {text-decoration: underline; text-align:center}
 P {font-Family:arial,verdana; color: white; background-color: black}
 BODY {color:black;background-color: #cccccc; text-indent:1cm}
 // -->
 </STYLE>
</head>
<body>
<h1>P&aacute;gina con estilos</h1>
Bienvenidos...
<p>Esto es un ejemplo</p>
</body>
</html>
```

Estilo definido para todo un sitio Web

De una vez podemos definir los estilos de todo un sitio Web. Esto se consigue creando un archivo donde tan sólo colocamos las declaraciones de estilos de la página y enlazando todas las páginas del sitio con ese archivo.

Veamos el proceso para incluir estilos con un archivo externo.

1- Creamos el archivo con la declaración de estilos. Es un fichero de texto normal con la extensión .css para aclararnos qué tipo de archivo es.

2- Enlazamos la página Web con la hoja de estilos. Para ello, vamos a colocar la etiqueta <LINK> con los atributos

- rel="STYLESHEET" indicando que el enlace es con una hoja de estilos
- type="text/css" porque el archivo es de texto, en sintaxis CSS
- href="estilos.css" indica el nombre del fichero fuente de los estilos

Sintaxis CSS

- Para definir un estilo se utilizan atributos como font-size,text-decoration... seguidos de dos puntos y el valor que le deseemos asignar. Podemos definir un estilo a base de definir muchos atributos separados por**;** font-size: 10pt; text-decoration: underline;
- Para definir el estilo de una etiqueta se escribe la etiqueta seguida de la lista de atributos encerrados entre llaves. Ejemplo: H1{text-align: center; color:black}

JAVASCRIPT

Se trata de un lenguaje de tipo script compacto, basado en objetos y guiado por eventos diseñado específicamente para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de Internet.

Los programas JavaScript van incrustados en los documentos HMTL, y se encargan de realizar acciones en el cliente, como pueden ser pedir datos, confirmaciones, mostrar mensajes, crear animaciones, comprobar campos...

Al ser un lenguaje de tipo script significa que no es un lenguaje compilado, es decir, tal cual se va leyendo se ejecuta por el cliente. Estar guiado por eventos significa que no vamos a tener un programa que se ejecute de principio a fin en cuanto carguemos la página Web. Significa que, cuando en el navegador suceda algún evento, entonces, si lo hemos decidido así, pasará ALGO. Y ese algo será alguna función JavaScript. Al ser

guiado por eventos, no tenemos una función principal que se ejecute por delante de las demás, sino que tendremos funciones, y, por ejemplo, si se pulsa el ratón sobre un cierto enlace, entonces se ejecutará una función, pero si se pulsa sobre una zona de una imagen sensible puede ejecutarse otra función.
El programa que va a interpretar los programas JavaScript es el propio explorador, lo que significa que si el nuestro no soporta JavaScript, no podremos ejecutar las funciones que programemos.

Lo primero que uno debe saber antes de programar nada en JavaScript, es cómo incrustarlo en un documento HTML. Para ello, tenemos dos formas; una, usando una directiva especial y otra, como parámetro de ciertas directivas (como <A>, <IMG>,...) en respuesta a eventos de usuario ya predefinidos (pulsar el ratón, cargarse la página,...).
El primer método consiste en usar la directiva pareada <SCRIPT>. Además, no hemos de olvidar que habrá quien tenga un explrador que no soporte JavaScript y habrá quien sí pero no quiera soportarlo, así que para prevenir que el código aparezca en las páginas encerraremos el código JavaScript entre comentarios:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
Código JavaScript
//-->
</SCRIPT>
```

En principio no importa mucho dónde pongamos el código, sin embargo, una buena costumbre es introducirlo en la cabecera del documento HTML (entre <HEAD> ...</HEAD>).
La otra forma de introducir código JavaScript en nuestros documentos HTML es en respuesta a determinados eventos que puede recibir el documento como consecuencia de las acciones del usuario que esté viendo la página en ese momento. Estos eventos están asociados a ciertas directivas HTML. La forma general sería esta: <DIRECTIVA nombreEvento="Codigo_JavaScript"> (ver más adelante tabla de eventos disponibles).

Sintaxis de JavaScript
El lenguaje presenta los siguientes elementos:

- Es "Case Sensitive", es decir, distingue mayúsculas de minúsculas.
- Comentarios: /* ... */ para encerrar un bloque y // para comentarios de una línea.
- Cada sentencia ha de terminar en **;**
- Encerrando código entre llaves { ... } lo agrupamos.

En JavaScript también podemos definir funciones (por medio de la palabra reservada function), la estructura general de la definición de una función es:

```
function Nombre_Funcion(arg1, ..., argN) {
        codigo de la funcion
        return valor;
}
```

Podremos llamar a nuestra función como respuesta a algún evento, por ejemplo:

```
<A HREF="doc.htm" onClick="Nombre_Funcion(arg1,...);">Pulse aquí</A>.
```

Los eventos más importantes que reconoce Javascript son:

onFocus	El usuario se posiciona en una ventana o cuadro de texto de un formulario
onKeyDown	Se pulsa una tecla
onKeyPress	Se pulsa o se libera una tecla
onKeyUp	Se libera una tecla
onLoad	Se carga un documento en el explorador
onMouseDown	Se pulsa un botón del ratón
onMouseMove	Se mueve el cursor
onMouseOver	El puntero del ratón se posiciona sobre un enlace
onMouseOut	El puntero del ratón sale de un enlace o imagen mapa
onMouseUp	Se libera un botón del ratón
onSelect	Se selecciona una de las opciones de un cuadro combo del formulario
onSubmit	Se pulsa el botón submit del formulario
unLoad	El usuario sale de la página

Capa de datos

El nivel de servicios de datos es responsable de:

- Almacenar los datos.
- Recuperar los datos.
- Mantener los datos.
- La integridad de los datos.

Los servicios de datos tienen una variedad de formas y tamaños, incluyendo los sistemas de administración de bases de datos relacionales (SABDs) como Microsoft SQL Server, servidores de correo electrónico como Microsoft Exchange Server y sistemas de archivos tales como el Sistema de Archivos NTFS.

MODELO DE DATOS

Un modelo de datos es un conjunto de conceptos que sirven para describir la estructura de una base de datos: los datos, las relaciones entre los datos y las restricciones que deben cumplirse sobre los datos. Los modelos de datos contienen también un conjunto de operaciones básicas para realizar consultas (lecturas) y actualizaciones de datos. Los modelos de datos más modernos incluyen conceptos para especificar comportamiento, permitiendo especificar un conjunto de operaciones definidas por el usuario.

Se pueden clasificar dependiendo de los tipos de conceptos que ofrecen para describir la estructura de la base de datos. Los modelos de datos de alto nivel, o modelos conceptuales, disponen de conceptos muy cercanos al modo en que la mayoría de los usuarios percibe los datos., mientras que los modelos de datos de bajo nivel, o modelos físicos, proporcionan conceptos que describen los detalles de cómo se almacenan los datos en la computadora.
Los conceptos de los modelos físicos están dirigidos al personal informático, no a los usuarios finales. Entre estos dos extremos se encuentran los modelos lógicos cuyos conceptos pueden ser entendidos por los usuarios finales aunque no están demasiado alejados de la forma en que los datos se organizan físicamente. Los modelos lógicos ocultan algunos detalles de cómo se almacenan los datos, pero pueden implementarse de manera directa en una computadora.

Los modelos conceptuales utilizan conceptos como entidades, atributos y relaciones. Una entidad representa un objeto o concepto del mundo real como, por ejemplo, un empleado de la empresa inmobiliaria u oficina. Un atributo representa alguna propiedad de interés de una entidad como, por ejemplo, el nombre o el salario del empleado. Una relación describe una interacción entre dos o más entidades, por ejemplo, la relación de trabajo entre un empleado y su oficina.

La metodología de diseño de datos divide cada modelo en tres esquemas:

a) MODELO GLOBAL: Se trata de una representación gráfica legible por el usuario y que nos aporta el flujo de información dentro de una organización. No existen reglas para su construcción y se debe realizar siempre el esquema más sencillo posible para la comprensión por parte del usuario de la base de datos.
b) MODELO LÓGICO: Se trata de una representación gráfica, mediante símbolos y signos normalizados de la base de datos. Su objetivo es representar la estructura de los datos y las dependencias de los mismos, garantizando la consistencia y evitando la duplicidad.
c) MODELO FÍSICO: Se trata del almacén de los datos, es la base de datos en sí misma, el soporte donde se almacenan los datos y de donde se extraen para convertir los datos en información. En función del gestor de base de datos empleado las reglas de almacenamiento varían.

CONSTRUCCIÓN DE UNA BASE DE DATOS
El ciclo de vida de una base de datos puede descomponerse en tres etapas:

1. CONCEPCIÓN. Consiste en reproducir el mundo real con ayuda de uno de los modelos de datos conocidos (relacional). El resultado de esta fase es un esquema escrito según un formalismo cualquiera no interpretado por el SGBD.
2. CREACIÓN DE LA BASE DE DATOS VACÍA: Consiste en traducir este esquema en órdenes comprensibles para el SGBD, como resultado se obtiene la estructura de la base de datos desprovista de cualquier tipo de información.
3. EXPLOTACIÓN: Aquí los registros serán manipulados con la ayuda del lenguaje de consultas (SQL). Es ahora cuando los usuarios pueden consultar los datos y ponerlos a punto durante el resto de la vida de la base.

En las tablas de debe procurar que no haya duplicidad de datos, conservar la integridad, etc.:

- Redundancia de datos
- Incoherencia de datos
- Pérdida de datos
- Estado de la tabla: Se dice que una tabla está en estado de 1ª forma normal si toda columna de esta tabla no puede tener más que valores atómicos, un valor es atómico si no es divisible.

Modelo Entidad-Relación

El **Modelo Entidad-Relación** es el modelo conceptual más utilizado para el diseño conceptual de bases de datos. Fue introducido por Peter Chen en 1976. Esta formado por un conjunto de conceptos que permiten describir la realidad mediante un conjunto de representaciones gráficas y lingüísticas.

Originalmente, el modelo entidad-relación sólo incluía los conceptos de entidad, relación y atributo. Más tarde, se añadieron otros conceptos, como los atributos compuestos y las jerarquías de generalización, en lo que se ha denominado modelo entidad-relación extendido.

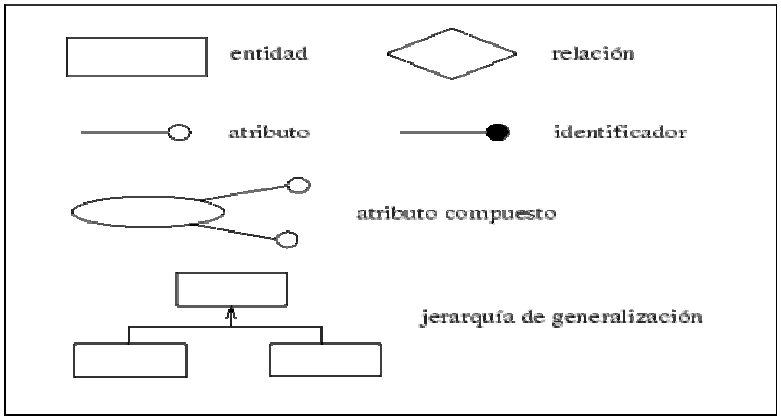


Figura 4.2. Componentes gráficos para crear el Modelo Entidad Relación.

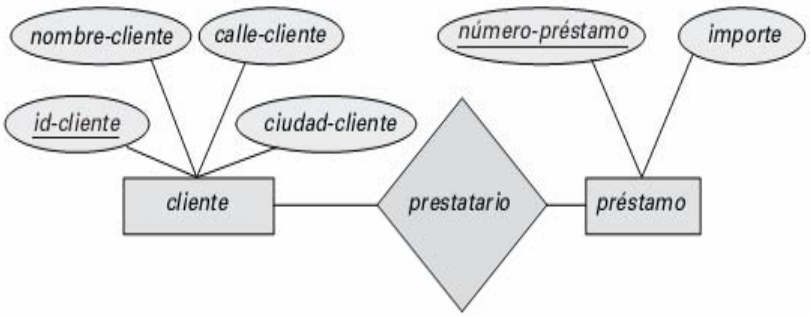


Figura 4.3. Ejemplo del Modelo Entidad Relación.

Metodología del diseño conceptual

1. Identificar las entidades.
2. Identificar las relaciones.
3. Identificar los atributos y asociarlos a entidades y relaciones.
4. Determinar los dominios de los atributos.
5. Determinar los identificadores.
6. Determinar las jerarquías de generalización (si las hay).
7. Dibujar el diagrama entidad-relación.
8. Revisar el esquema conceptual local con el usuario.

Normalización

La normalización se utiliza para mejorar el esquema conceptual de una base de datos, de modo que éste satisfaga ciertas restricciones que evitan la duplicidad de datos. Garantiza que el esquema resultante está más próximo al modelo de la empresa, es consistente, tiene la mínima redundancia y la máxima estabilidad.

Las bases de datos relacionales se normalizan para:

- Evitar la redundancia de los datos.
- Prevenir Problemas de actualización.
- Proteger la integridad de los datos.

Las primeras tres formas normales son suficientes para cubrir las necesidades de las mayorías de las bases de datos

- *Primera forma normal (1FN).* Se eliminan los grupos repetitivos. Los atributos han de ser atómicos. Gráficamente las celdas de la tabla contienen solo un valor, en cada uno de los atributos sólo se puede incluir un dato, aunque sea compuesto, pero no se pueden incluir una lista de datos. Se trata de de que cada atributo guarde la menor cantidad de información posible.

- *Segunda forma normal (2FN).* Una tabla está en Segunda Forma Normal o 2FN cuando está en 1FN y todo atributo que no pertenece a la clave primaria tiene una dependencia funcional de la clave completa y no de parte de ella.

- *Tercera forma normal (3FN).* Se eliminan las dependencias funcionales transitivas. Se dice que hay dependencia funcional transitiva entre dos atributos cuando un atributo que no pertenece a la clave primaria permite conocer el valor de otro atributo. Para que una tabla esté en 3FN tiene que estar previamente en 2FN.

El diseño de bases de datos consta de tres etapas:

- Diseño Conceptual
- Diseño Lógico
- Diseño Físico

El diseño lógico es el proceso mediante el que se construye un esquema que representa la información que maneja una empresa, basándose en un modelo lógico determinado, pero independientemente del SGBD concreto que se vaya a utilizar para implementar la base de datos e independientemente de cualquier otra consideración física.
Algo importante es la conversión del esquema conceptual a un esquema lógico adecuado al modelo relacional. Para ello, se debe:

- Eliminar las relaciones de muchos a muchos
- Eliminar las relaciones complejas
- Eliminar las relaciones recursivas
- Eliminar las relaciones con atributos
- Eliminar los atributos multivaluados
- Reconsiderar las relaciones de uno a uno
- Eliminar las relaciones redundantes.

Conectividad a Bases de Datos con Java (API JDBC)

El JDBC (Java Database Conectivity) es el estándar para conectar el lenguaje Java y una amplia gama de Sistemas Manejadores de Bases de Datos (DBMS). Proporciona un puente de llamada para el acceso de bases de datos basados en SQL.

El API JDBC permite:

- Establecer una conexión con una base de datos
- Ejecutar sentencias SQL
- Procesar los resultados de las consultas

El JDBC provee un conjunto de clases con interfase genérica, que son implementados por un controlador de base de datos específico. El API provee acceso a cualquier base de datos relacional que tenga soporte para JDBC.
El uso de la base de datos consta de cinco paso:

1. Cargar el driver. Es muy sencillo y sólo implica una línea de código (no sin antes colocar el driver en su directorio lib de su aplicación web) así: ClassforName("com.mysql.jdbc.Driver"); También se define la URL, que es el protocolo jdbc:servidor/puerto/base/usuario/password

2. Establecer la conexión. La conexión se realiza con getConnection: Connection conexion=DriverManager.getConnection(url,"micka","xxx");
3. Realizar la consulta a la BD. Se crea el objeto Statement, el cual es usado para enviar consultas SQL utilizando el método executeQuery, el cual regresa un objeto de tipo ResultSet.
   String consulta="Select columna1 FROM tabla1";
   ResulSet rs= statement.executeQuery(consulta);
4. Extraer los resultados. Con el siguiente código se accede a los valores almacenados en la fila actual de rs e imprime una línea con el valor de columna1. Cada vez que el método next es llamado, la siguiente fila se convierte en la actual, y así hasta que no haya más filas en rs.
   While(rs.next()) { String s= rs.getString("nombre");
   System.out.println(s); }
5. Cerrar las conexiones. Se hace con la instrucción conexion.close() .

Reglas del negocio

Toda aplicación trata de reflejar parte del funcionamiento del mundo real. Para ello, es necesario que cada aplicación refleje las restricciones que existen en el negocio dado, de modo que nunca sea posible llevar a cabo acciones no válidas. A las reglas que debe seguir la aplicación para garantizar esto se las llama *reglas de negocio*, o *business rules*. Ejemplos de tales reglas son: no permitir crear facturas pertenecientes a clientes inexistentes, controlar que el saldo negativo de un cliente nunca sobrepase cierta cantidad, etc. Los servicios de negocios son el "puente" entre un usuario y los servicios de datos.

El nivel de servicios de negocios es responsable de:

- Recibir la entrada del nivel de presentación.
- Interactuar con los servicios de datos para ejecutar las operaciones de negocios para los que la aplicación fue diseñada a automatizar (por ejemplo, la preparación de impuestos por ingresos, el procesamiento de ordenes y así sucesivamente).
- Enviar el resultado procesado al nivel de presentación.

SERVLETS

Los servlets son módulos que permiten sustituir o utilizar el lenguaje java en lugar de los CGI realizados con otros lenguajes como C++, Perl. Los CGI son aplicaciones que se ejecutan en un servidor Web en respuesta a una acción de un browser remoto (petición de una página HTML, envío de datos de un formulario, etc.). Permiten generar páginas HTML dinámicas, esto es páginas HTML cuyo contenido puede variar y que por lo tanto no pueden manejarse en un archivo en el servidor. Los servlets no tienen entorno gráfico ya que se ejecutan en el servidor. Reciben unos datos y su salida o respuesta son principalmente archivos de texto HTML. Los servlets son desarrollados utilizando el API Java Servlet, que es una extensión de Java que hasta la fecha no forma parte de ninguno de los JDK. Es necesario instalar el software específico de Java Servlet.

Características

- Son independientes del servidor utilizado y de su sistema operativo.
- Los servlets pueden llamar a otros servlets, e incluso métodos concretos de otros servlets. De esta forma se puede distribuir de forma más eficiente el trabajo a realizar. Permiten redireccionar peticiones de servicios a otros servlets (en la misma máquina o en una máquina remota).
- Pueden obtener fácilmente información acerca del cliente (la permitida por el protocolo http), tal como su dirección IP, el puerto que se utiliza en la llamada, el método utilizado (GET, POST,…), etc.
- Permiten además la utilización de cookies[1] y sesiones de forma que se puede guardar información específica acerca de un usuario determinado, personalizando de esta forma la interacción cliente-servidor. Una clara aplicación es mantener la sesión con un cliente.

[1] Una *cookie* es un fragmento de información que se almacena en el disco duro del visitante de una página web a través de su explorador, a petición del servidor de la página. Esta información puede ser luego recuperada por el servidor en posteriores visitas.

- Los servlets pueden actuar como enlace entre el cliente y una o varias bases de datos en Arquitectura cliente-servidor de tres capas (si la base de datos está en un servidor distinto).
- Puede realizar tareas de Proxy[2] para un applet[3]. Debido a las restricciones de seguridad un applet no puede acceder directamente por ejemplo a un servidor de datos localizado en cualquier máquina remota, pero el servlet si puede hacerlo de su parte.
- Permiten la generación dinámica de código HTML dentro de una propia página HTML. Así pueden usarse servlets para la creación de contadores, banners, etc.
- Para crear un servlet se necesita heredar de la clase HttpServlet del paquete servlet. Así: public class MiServlet extendí HttpServlet {}

JavaServer Pages (JSP)

Una página JSP es básicamente una página Web con HTML tradicional y código Java incrustado. La extensión de fichero de una página JSP es ".jsp" en vez de ".html" o ".htm", y eso le dice al servidor que esta página requiere un manejo especial que se conseguirá con una extensión del servidor o un plug-in. Aquí hay un sencillo ejemplo:

date.jsp

```
<HTML>
<BODY BGCOLOR="ffffcc">
<H2>Fecha y hora</H2>
        <%
        java.util.Date today = new java.util.Date();
        out.println("Hoy es: "+today);
        %>
</BODY>
</HTML>
```

Este ejemplo contiene HTML tradicional y algún código Java. La etiqueta <% identifica el inicio de un scriptlet, y la etiqueta %> identifica el final de un scriptlet. Cuando se llame a ésta página (date.jsp), será compilada (por el motor JSP) en un Servlet Java. En este momento el Servlet es manejado por el motor Servelt como cualquier otro Servlet. El motor Servlet carga la clase Servlet (usando un cargador de clases) y lo ejecuta para crear HTML dinámico para enviarlo al explorador.

La siguiente vez que se solicite la página, el motor JSP ejecuta el Servlet ya cargado a menos que la página JSP haya cambiado, en cuyo caso es automáticamente recompilada en un Servlet y ejecutada.

[2] Un *applet* es un componente de *software* que corre en el contexto de otro programa, por ejemplo un explorador web. A diferencia de un *programa*, un *applet* no puede correr de manera independiente, ofrece información gráfica y a veces interactúa con el usuario, típicamente carece de sesión y tiene privilegios de seguridad restringidos.

[3] El término proxy hace referencia a un programa o dispositivo que realiza una acción en representación de otro. La finalidad más habitual es la del servidor proxy, que sirve para permitir el acceso a Internet a todos los equipos de una organización cuando sólo se puede disponer de un único equipo conectado, esto es, una única dirección IP.

El código fuente de una página JSP puede contener:

- Directivas: Indican información general de la página, como puede ser importación de clases, página a invocar ante errores, si la página forma parte de una sesión, etc.
- Declaraciones: Sirven para declarar métodos o variables.
- Scriptlets: Código Java embebido.
- Expresiones: Expresiones Java que se evalúan y se envían a la salida.
- Tags JSP: Etiquetas especiales que interpreta el servidor.

Directivas Las directivas son elementos que proporcionan información al motor JSP.
Hay tres tipos de directivas:

- Page: Se indica con la forma <%@ page atributo="valor">. Tiene diversos usos, entre los cuales destacaremos: - Importar clases. Importar código, de la misma forma que se realiza en un programa en Java, se indica con el atributo import. Ejemplo: <%@page import="java.io.*, miPackage.miClase"%> - Indicar si la página tendrá acceso a la sesión

- Include: Permite incluir un archivo en el lugar donde se especifique, simplemente copia el contenido del archivo byte a byte, siendo el resultado similar a si copiáramos el texto del archivo incluido y lo pegáramos en el JSP. Ejemplo:

```
<html>
    <head> <%@ include file="titulo.txt"%> </head>
    <body> <%@ include file="cuerpoPagina.jsp"%>
    </body>
</html>
```

- Taglib: Se emplea para indicar que se van a emplear librerías de etiquetas. Ejemplo: <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

Código Java Podemos insertar código Java dentro de JSP de tres formas:

- Expresiones: Son fragmentos de código Java, con la forma <%= expresión %> que se evalúan y se muestran en la salida del explorador. Ejemplo: <%= "Tamaño de cadena: "+cadena.length() %>

- *Scriptlets:* Son fragmentos de código Java con la forma <% código %>, en general, podemos insertar cualquier código que pudiéramos usar dentro de una función Java. Ejemplo:

```
<table>
    <% for (int i=0;i<10;i++) { %>
    <tr><td> <%=i%>
    </td></tr> <% } %>
</table>
```

- Declaraciones: Contienen declaraciones de variables o métodos, con la forma <%! declaración %>. Estas variables o métodos serán accesibles desde cualquier lugar de la página JSP. Ejemplos: <%! int numeroAccesos=0; %>

```
<html>
    <body>
        <%= "La página ha sido accedida "+(++numeroAccesos)+ " veces desde el arranque del servidor" %>
    </body>
</html>
```

Procesado de formularios HTML con JSP

Una de las partes más comunes en aplicaciones de comercio electrónico es un formulario HTML donde el usuario introduce alguna información como su nombre y dirección. Usando JSP, los datos del formulario (la información que el usuario introduce en él) se almacenan en un objeto request que es enviado desde el explorador hasta el contenedor JSP. La petición es procesada y el resultado se envía a través de un objeto response de vuelta al explorador. Estos dos objetos están disponibles implícitamente para nosotros.

Componentes Reutilizables

El ejemplo del formulario anterior es simple en el sentido de que no hay mucho código implicado. Cuanto más código esté implicado, más importante es no mezclar la lógica del negocio con la presentación final en el mismo archivo. La separación de la lógica de negocio de la presentación permite cambios en cualquier sitio sin afectar al otro. Sin embargo, el código de producción JSP se debe limitar a la presentación final. Así pues, ¿cómo implementamos la parte de la lógica de negocio?

Aquí es donde los **JavaBeans** entran en juego. Esta tecnología es un modelo de componente portable, independiente de la plataforma que permite a los desarrolladores escribir componentes y reutilizarlos en cualquier lugar. En el contexto de JSP, los JavaBeans contienen la lógica de negocio que devuelve datos a un script en una página JSP, que a su vez formatea los datos devueltos por el componente JavaBean para su visualización en el explorador. Una página JSP utiliza un componente JavaBean fijando y obteniendo las propiedades que proporciona.

Hay muchos beneficios en la utilización de JavaBeans para mejorar las páginas JSP:

- Componentes Reutilizables: diferentes aplicaciones pueden reutilizar los mismos componentes.
- Separación de la lógica de negocio de la lógica de presentación: podemos
- modificar la forma de mostrar los datos sin que afecte a la lógica del negocio.
- Proteger el código fuente.

Como ejemplo supongamos que creamos un formulario con dos campos: nombre y email. En JavaBeans, los campos del formulario son llamados propiedades. Por eso, primero escribimos un componente JavaBean con métodos setX getX, donde X es el nombre de la propiedad. Por ejemplo, si tenemos unos métodos llamados setNom y getNom entonces tenemos una propiedad llamada nom. Para conseguir la reutilización del componente, debemos seguir dos reglas importantes:

- Nuestra clase bean debe proporcionar un constructor sin argumentos para que pueda ser creado usando Beans.instantiate.

- Nuestra clase bean debe soportar persistencia implementando el interface Serializable o Externalizable.

El ejemplo de abajo muestra un componente FormBean.

FormBean.java

```
import java.io.*;
public class FormBean implements Serializable {
        private String nom;
        private String email;
        public FormBean() {
                nom = null;
                email = null;}
        public void setNom(String nom) {
                this.nom = nom;}
        public String getNom() {
                return nom;}
        public void setEmail(String email) {
                this.email = email;}
        public String getEmail() {
                return email;}
}
```

Para poder usar el componente FormBean en el archivo JSP de nuestro formulario, necesitamos ejemplarizar el componente. Esto se hace usando la etiqueta <jsp:useBean>. La siguiente línea <jsp:setProperty> se ejecuta cuando se ha ejemplarizado el bean, y se usa para inicializar sus propiedades. En este caso, ambas propiedades (nom y email) se configuran usando una sola sentencia.

```
<jsp:useBean id="formbean" class="userinfo.FormBean"/>
<jsp:setProperty name="formbean" property="*"/>
<HTML>
<BODY BGCOLOR="#ffffcc">
<% if (request.getParameter("nom")==null && request.getParameter("email") == null) { %>
<form method="GET" action="process2.jsp">
Nombre: <input type="text" name="nom" size=27>
email: <input type="text" name="email" size=27>
<input type="submit" value="Process">
</FORM>
<% } else { %>
<B>Tu información:</B><P>
<B>Nombre</B>: <jsp:getProperty name="formbean" property="name"/><P>
<B>Email</B>: <jsp:getProperty name="formbean" property="email"/><% } %></BODY></HTML>
```

Arquitectura del framework Struts

Un framework Web, podemos definirlo como un conjunto de componentes (por ejemplo clases en java, descriptores y archivos de configuración en XML, entre otros) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web.

Existen varios tipos de frameworks Web: orientados a la interfaz de usuario, como Java Server Faces, orientados a aplicaciones de publicación de documentos, como Coocon, orientados a la parte de control de eventos, como Struts que ofrece su propio componente controlador y proporciona integración con otras tecnologías para implementar el modelo, mediante tecnologías de acceso a datos como JDBC, y la vista mediante JSP.

El framework Struts es la implementación del patrón MVC en aplicaciones Web. Es una arquitectura completamente estructurada que divide perfectamente lógica de negocio (Model), presentación (View) y control de flujo de aplicaciones(Controller).
Los Struts realmente proveen un conjunto de clases y TAG-LIBS que conforman el Controlador, la integración con el Modelo (o lógica de negocio) y facilitan la construcción de vistas.

¿Que proporciona struts?

- Un servlet (ActionServlet) que actúa como controlador MVC totalmente configurable.
- Clases base que son extendidas para implementar la lógica de la aplicación web:
  - Struts Action
  - Struts ActionForm
- Un rico conjunto de etiquetas personalizadas JSP que cooperan con el controlador para su uso en la capa Vista de MVC.
- Varias opciones para la validación de entrada de usuario en formularios HTML: ActionForm o Validator Framework.
- Mecanismos para el manejo y reporte de errores.
- Un archivo de configuración: struts-config.xml. Ahí se especifican todas las relaciones entre acciones y clases, formularios y clases, acciones y jsp de presentación, que globalmente conforman el “mapa” de la aplicación.

A continuación se explican con mayor detalle cada uno de los componetes de Struts:

Actions

- Se crea una acción extendiendo la clase org.apache.struts.action.Action.
- El ActionServlet ejecuta acciones invocando el método execute() .
- Dentro del método execute() se tiene acceso a:
  - Cabeceras y parámetros de peticiones HTTP
  - Atributos/beans guardados en los contextos application/session/request scope.
  - Struts ActionForm asociados con la acción (opcional).
  - El ActionMapping associado a esta acción (opcional).
  - El objeto httpResponse

- El método execute() devuelve un objeto ActionForward que indica al ActionServlet a dónde transferir el control a continuación.

Uso del DispatchAction (Despachador de Destinos)

En algunos casos es necesario que en un mismo Action se realicen diferentes acciones o procesos, es entonces cuando se hace uso del DispatchAction, ahí se definen todos los métodos que se requieran y se identifica el tipo de acción a realizar a través del uso de una variable parámetro, que se asigna en el struts-config al mapping correspondiente.
A diferencia de un Action normal, ahora nuestro Action debe heredar de DispatchAction.
Entre las ventajas de usar DispatchAction se encuentran:

- Agrupación de los Actions que están relacionados.
- Generar un código más modular.
- Evitar código redundante.

Form Beans

- Un ActionForm es un JavaBean con propiedades que corresponden a los controles de un formulario HTML. Los parámetros son mapeados a propiedades del bean.
- El programador define un formbean extendiendo la clase org.apache.struts.action.ActionForm.
- Hay que definir cada una de las propiedades en la clase y escribir los métodos de acceso (getters/setters) correspondientes, siguiendo las reglas de JavaBeans.
- Después de escribir el código del form bean, es necesario asociarlo con una o más acciones a través del fichero de configuración de Struts struts-config.xml
- Cada vez que se llama a la acción , el ActionServlet poblará las propiedades con los valores de los parámetros recibidos en el formulario HTML.

Struts Tag Libraries

- La framework Struts proporciona un conjunto de seis librerías de etiquetas, que asisten en la tarea de la creación de la vista de MVC para evitar incluir código Java en los JSPs:
    - Bean Tags
    - HTML Tags: Usadas para crear formularios de entrada de datos.
    - Logic Tags: Iteraciones, flujo de aplicación, etc.
    - Nested Tags
    - Template Tags
    - Tiles Tags

Tiles Tags: Uso de plantillas (tiles) en Struts

- La librería de etiquetas Tiles es un super-conjunto de la librería Templates.
- Intenta evitar la duplicación de contenido de lenguaje de marcado dentro de una aplicación web.
- Tiles reduce el tamaño de código redundante en una aplicación web y separa el contenido de la visualización del mismo de manera más eficiente
- En Struts, se configuran en el archivo tiles-defs.xml.

Cuando construimos un web, podemos considerar que éste está formado por una o varias plantillas en la que únicamente cada página cambia una porción (normalmente central) del contenido mostrado respecto a las restantes. La plantilla normalmente contiene un menú, publicidad, cabeceras, pies, etc.

De este modo, para hacer crecer el Web solo necesitamos ir añadiendo nuevas porciones. Además, cambiar todo el "look and feel" sería relativamente rápido.

La construcción de páginas en base a plantillas se basa en la construcción de tres elementos:

- La plantilla que define elementos estáticos y áreas donde insertar otros contenidos
- La página real que utiliza la plantilla insertando textos y otras porciones dinámicamente.
- Las porciones particulares reutilizables (que pueden ser otros JSPs)

Validator

Struts está integrado con el **Framework de validación Validator**, de Jakarta, permitiéndonos tener un mecanismo alternativo al de ActionForm para facilitar las acciones de validación de entrada. Su principal ventaja es que las validaciones no se escriben junto al código, sino que se configuran en los ficheros validation.xml y validator-rules.xml.

El Validator permite realizar las siguientes operaciones:

- Definir métodos de validación genéricos que puedan ser utilizados en diferentes formularios.
- Escribir las reglas de validación de los formularios en archivos de configuración (XML), de forma que para cualquier modificación sólo es necesario cambiar estos archivos y no el código de la aplicación.
- Utilizando los dos archivos de configuración (XML) antes mencionados, y sin necesidad de realizar cambios grandes, generar código que realice las mismas validaciones en el lado del cliente (Javascript).

El validador requiere dos archivos de configuración:

- **validator-rules.xml** es el archivo del framework donde se definen las reglas genéricas de validación existentes (tales como required, minlength, creditCard, email...) y a cada una se le asigna un nombre.

- **validation.xml** es el archivo en el que se indica de forma más concreta a qué formularios y campos se van a aplicar ciertas reglas de validación.

Flujo de control en struts

- La clase org.apache.struts.action.ActionServlet es el eje de Struts. Dada una petición de entrada HTTP:
  - Crea un objeto ActionForm donde guarda y valida los parámetros de entrada
  - Decide que objeto Action se debe invocar y le pasa el objeto ActionForm creado
  - Transfiere control a la siguiente etapa de procesamiento de la petición (típicamente un JSP).
- El fichero de configuración web.xml contiene los url mappings para enviar las peticiones de llegada al ActionServlet, mientras que el fichero de configuración de Struts struts-config.xml contiene los mappings a acciones
- Los form beans creados por ActionServlet deben ser implementados por el programador, extendiendo org.apache.struts.action.ActionForm.
  - El programador deberá definir un conjunto de getters y setter y sobreescribir los métodos validate() y reset()
- Los objetos Action invocados deben ser desarrollados por el programador y extienden org.apache.struts.action.Action. Tienen un método execute() o (perform() en Struts 1.0) que ejecuta la lógica de negocio
- La acción devuelve un objeto ActionForward al servlet que especifica el siguiente paso a ejecutar, normalmente se transfiere el control a un JSP para que visualice los resultados.

Pool de Conexiones en Struts

Si buscamos dónde consume mayor tiempo nuestra aplicación en la mayoría de los casos terminamos en el acceso a base de datos.

Por ello es uno de los puntos más importantes a tener en cuenta a la hora de reducir tiempos para mejorar el rendimiento de la aplicación. Uno de los elementos clave a este respecto, es el uso de un pool de conexiones.

Un pool de conexiones es un servicio que mantiene un grupo de conexiones a una base de datos abiertas de modo que cuando se le pide una conexión al pool este no necesariamente abre una si no que entrega una abierta, cuando cierras la conexión no lo haces físicamente sino que la devuelves al pool para que vuelva a usarla con otro que haga una petición, de este modo se reduce el enorme tiempo de abrir y cerrar conexiones.

Los servicios de pool de conexión no tienen que ver necesariamente con un servidor web, algunos servidores de aplicaciones traen este servicio disponible, como por ejemplo tomcat (el cual usaremos en nuestro sistema).

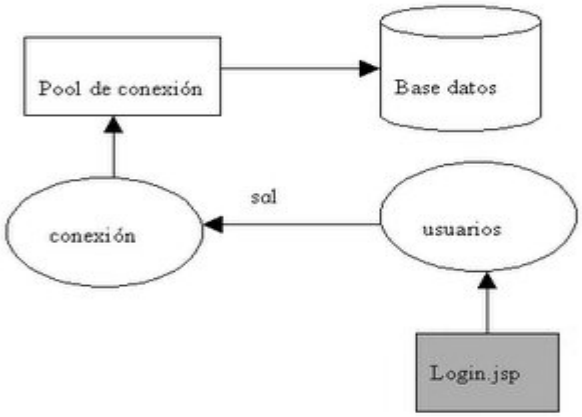


Figura 4.4. Esquema del pool de conexiones a una base de datos.

Configuración del Pool de Conexiones en Struts

1. Para configurar el pool, buscamos el archivo context.xml, ubicado dentro de la carpeta META-INF de nuestro proyecto. Indicamos la configuración para el pool, colocando el nombre para el pool, el driver jdbc, la base datos, el usuario y password.

```
<Context path="/<<nombre proyecto>>">
        <Resource name="jdbc/<<nombre>> " auth="Container"
      type="javax.sql.DataSource" maxActive="-1" maxIdle="-1" maxWait="-1"
      username="root" password="root"
      driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost:3306/<<nombre>>?autoReconnect=true"/>
</Context>
```

2. Configurar el archivo web.xml, para permitir a nuestras clases java, usar el pool. Añadimos en resource-ref.

```
<resource-ref>
      <res-ref-name> jdbc/<<nombre>> </res-ref-name>
      <res-type> javax.sql.DataSource </res-type>
      <res-auth> Container </res-auth>
 </resource-ref>
```

3. Clases que usan el pool. Definimos una clase que se encargue de manejar la conexión con el pool, la cual debe contener básicamente las siguientes lìneas para obtener la conexión.

```
Context init = new InitialContext();
DataSource ds = (DataSource) init.lookup("java:comp/env/jdbc/<<nombre>>");
cn = ds.getConnection();
```

V. IMPLEMENTACIÓN Y EVALUACIÓN DEL SISTEMA WEB (VIDEOCLUB)

Creación de la aplicación Videoclub en Struts

Para implementar nuestro sistema en el framework Struts, cada una de las actividades o procesos que se pueden realizar utilizan de manera general lo siguiente:

1. Una clase **Action** encargada de cargar el formulario.
2. Un **Bean** tipo **Form** cuyos campos o atributos de instancia coinciden con los campos del formulario. (al ser tipo JavaBean tiene los correspondientes métodos set y get para los atributos).
3. Una **JSP** que contiene el formulario citado. Los campos coinciden con los definidos en cada Bean de formulario.
4. Una clase **Action** encargada de pasar la instancia del formulario al Bean de Lógica de Negocio, esperar su resultado y redirigir el flujo a una JSP de error o de éxito.
5. Un **Bean** de **lógica** de negocio (El que hace algo con los datos del formulario).
6. Una clase **Service**, que no es un componente propio de Struts pero que es de utilidad para enlazar capa de negocio con los accesos a datos en la base.
7. Una clase **DAO** que permite el acceso a la información contenida en la base de datos.
8. Una **JSP** para mostrar un mensaje en caso de éxito.
9. Una **JSP** para mostrar los errores en caso de error.

Struts permite cierta flexibilidad en cuanto a nombres, pero por estandarización se especificó cual es Action, cual es Form, cual es Bean de negocio, etc, colocándoles su terminación correspondiente, por ejemplo AltaSocioAction. Por la misma razón fueron almacenados en subpaquetes que identifican a los componentes: com.diplomado.videoclub.Actions.

A continuación se presentan las porciones más representativas de los **archivos de configuración** en Struts del proyecto.

De los **ARCHIVOS DE CONFIGURACIÓN**, el más importante es el struts-config.xml, ya que ahí se mapea todo las posibles acciones del sistema y las rutas que deben de seguirse.
En la primera parte se definen todos los FormBeans que componen el sistema. Posteriormente se encuentra el mapping de los Actions. Ésta es una porción de los FormBeans y Actions más representativos.

struts-config.xml

```
<struts-config>
  <form-beans>
        <form-bean name="RentaForm" type="com.diplomado.videoclub.Forms.RentaForm"/>
        <form-bean name="PeliculaForm" type="com.diplomado.videoclub.Forms.PeliculaForm"/>
        <form-bean name="SocioForm" type="com.diplomado.videoclub.Forms.SocioForm"/>
        <form-bean name="LoginForm" type="com.diplomado.videoclub.Forms.LoginForm"/>
        <form-bean ..............
  </form-beans>

  <action-mappings>
        <action path="/Login" forward=".Login"/>
        <action path="/AltaSocio" forward=".AltaSocio"/>
        <action path="/ConsultaPelicula" forward=".ConsultaPelicula"/>
        <action path="/RegistraRenta" forward=".RegistraRenta"/>
        <action path ..............
        <action  input="/" name="LoginForm" path="/cambiaPassword" scope="session"
                 type="com.diplomado.videoclub.Actions.CambiaPassword"/>
        <action  input="/"  name="RentaForm"  path="/renta"  scope="session"
                  parameter= "method"
                  type="com.diplomado.videoclub.Actions.RentaAction">
                   <forward name="baja" path=".EditaSocio"/>
                   <forward name="trae" path=".AltaRenta"/>
        </action>

        <action  input="/"  name="AltaSocioForm"  path="/altaSocio"  scope="request"
                 parameter="method"
                 type="com.diplomado.videoclub.Actions.AltaSocioAction">
                <forward name="alta" path=".ExitoAltaSocio"/>
                <forward name="trae" path="/ENCARGADO/alta_socio.jsp"/>
        </action>
        <action  input="/"  name="LoginForm"  path="/checaLogin"  scope="session"
                 type="com.diplomado.videoclub.Actions.LoginAction">
                 <forward name="Acceso_Encargado" path=".Menu"/>
                 <forward name="Acceso_Denegado" path=".AccesoDenegado"/>
        </action>
        <action   input="/"  name="RenuevaCredencialForm"   path="/renuevaCredencial"
                   scope="session"   parameter= "method"
                   type="com.diplomado.videoclub.Actions.RenuevaCredencialAction">
                  <forward name="imprime" path=".ImprimeCredencial"/>
                   <forward name="trae" path=".ExitoAltaSocio"/>
        </action>
        <action ..................
  </action-mappings>
</struts-config>
```

El archivo validation.xml contiene las validaciones que se hacen del lado del cliente (javascript) a todos los formularios antes de ser enviados al servidor, utilizando las facilidades que brinda el framework Validator de struts. Aquí se muestra una porción del archivo, donde se encuentra la definición de la validación del formulario para la autentificación del usuario. Donde se especifica el nombre del Form específico y cada uno de los campos con sus respectivas validaciones: si el campo es requerido, o sólo permite ciertos caracteres, etc.

validation.xml

```
<form-validation>

  <formset>

    <form name="LoginForm">
      <field
        property="txtUsuario"
        depends="required, mask">
          <arg0 key="form.usuario"/>
          <var>
            <var-name>mask</var-name>
            <var-value>^[0-9a-zA-Z]*$</var-value>
          </var>
      </field>
      <field
        property="txtPassword"
        depends="required, mask">
          <arg0 key="form.password"/>
          <var>
            <var-name>mask</var-name>
            <var-value>^[0-9a-zA-Z]*$</var-value>
          </var>
      </field>
    </form>
  </formset>

</form-validation>
```

El documento tiles-defs.xml, es utilizado en nuestra aplicación para permitir una mejor maquetación de las páginas y no repetir código en las JSP's, ya que como se verá en la interfaz, el menú que se encuentra en la parte superior de las mismas, siempre está fijo. Aquí definimos la estructura visual de toda la aplicación, es decir la maqueta principal que contendrá a todas las páginas, y, que son requeridas en el struts-config.xml, refiriendo a cada nombre de los definitions tiles, por ejemplo <action path="/Login" forward=".Login /> que llama al primer definition de tiles.

tiles-defs.xml

```
<tiles-definitions>
  <definition name=".Login" path="/ENCARGADO/login.jsp">
  </definition>
  <definition name=".Principal" path="/maqueta.jsp">
    <put name="cuerpo" value="ENCARGADO/menu.jsp"/>
  </definition>
  <definition name=".Menu" path="/maqueta.jsp">
    <put name="imagen" value="IMG/logo2.bmp"/>
    <put name="imageng" value="IMG/grande1.bmp"/>
    <put name="cuerpo" value="ENCARGADO/menu.jsp"/>
  </definition>
  <definition name=".AccesoDenegado" path="/error.jsp">
    <put name="error" value="ENCARGADO/acceso_denegado.jsp"/>
  </definition>
  <definition name=".AltaSocio" path="/maqueta1.jsp">
    <put name="menu" value="ENCARGADO/menu.jsp"/>
    <put name="cuerpo" value="altaSocio.do?method=traeEdos"/>
  </definition>
  <definition name=".ConsultaSocio" path="/maqueta1.jsp">
    <put name="menu" value="ENCARGADO/menu.jsp"/>
    <put name="cuerpo" value="ENCARGADO/consulta_socio.jsp"/>
  </definition>
  <definition name=".EditaSocio" path="/maqueta1.jsp">
    <put name="menu" value="ENCARGADO/menu.jsp"/>
    <put name="cuerpo" value="ENCARGADO/edita_socio.jsp"/>
  </definition>
  <definition name=".AltaPelicula" path="/maqueta1.jsp">
    <put name="menu" value="ENCARGADO/menu.jsp"/>
    <put name="cuerpo" value="ENCARGADO/alta_pelicula.jsp"/>
  </definition>      <definition name=".BajaPelicula" path="/maqueta1.jsp">
    <put name="menu" value="ENCARGADO/menu.jsp"/>
    <put name="cuerpo" value="ENCARGADO/baja_pelicula.jsp"/>
  </definition>
  <definition name=".ConsultaPelicula" path="/maqueta1.jsp">
    <put name="menu" value="ENCARGADO/menu.jsp"/>
    <put name="cuerpo" value="ENCARGADO/consulta_pelicula.jsp"/>
  </definition>
  <definition name=".RegistraRenta" path="/maqueta1.jsp">
    <put name="menu" value="ENCARGADO/menu.jsp"/>
    <put name="cuerpo" value="ENCARGADO/registra_renta.jsp"/>
  </definition>
</tiles-definitions>
```

Los dos archivos siguientes, son utilizados básicamente para definir nuestro pool de conexiones que se usa en la aplicación para acceder a la base de datos “videoclub” creada en MySQL, aprovechando las posibilidades que brinda struts para la configuración del pool a través de éstos dos archivos (web.xml y context.xml). Las porciones que se visualizan aquí de ellos, son referentes a este punto:

web.xml

```
.................
     <resource-ref>
        <res-ref-name>jdbc/videoclub</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
      </resource-ref>
   </web-app>
```

context.xml

```
Context path="/videoclub">
<Resource name="jdbc/videoclub" auth="Container"
type="javax.sql.DataSource" maxActive="-1" maxIdle="-1" maxWait="-1"
        username="root" password="root"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://localhost:3306/videoclub?autoReconnect=true"/>
</Context>
```

LA BASE DE DATOS

Ahora pasemos a una parte de gran relevancia en nuestra aplicación, la base de datos. Nuestra base de datos fue creada en MySQL. He aquí nuestro diagrama relacional, el cual complementa el diagrama que se muestra en el capítulo II, cuando se crearon las clases entidad.

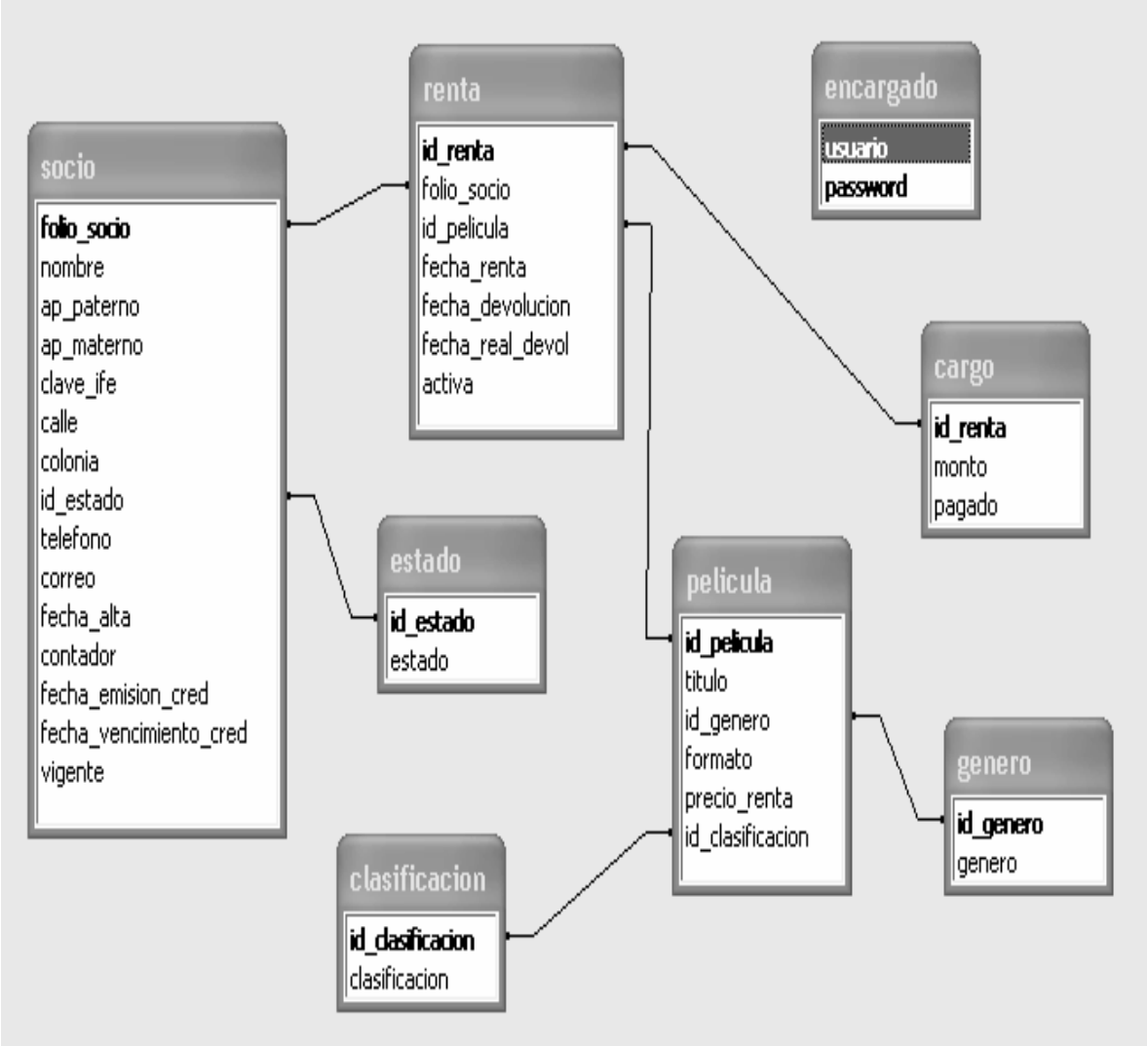


Figura 5.1. Diagrama relacional de la base de datos “videoclub”.

A continuación daremos una descripción breve de cada una de las tablas y campos que componen a la base como referencia de su estructura final para ser generadas en MySQL.

Diccionario de Datos

Tabla 1

SOCIO		
Campo	**Tipo**	**Descripción**
folio_socio	int	El identificador único del socio, que es un número generado automáticamente
nombre	varchar	El nombre del socio
ap_paterno	varchar	El apellido paterno del socio
ap_materno	varchar	El apellido materno del socio
clave_ife	varchar	La clave de la credencial de elector (IFE)
calle	varchar	Nombre de la calle
colonia	varchar	Nombre de la colonia
id_estado	int	El estado donde reside el socio
telefono	varchar	El teléfono particular del socio
correo	varchar	Una dirección de correo electrónico del socio
fecha_alta	date	Fecha en que se ha dado de alta
contador	int	Un contador para llevar un control de el número de películas rentadas (máximo 5)
fecha_emision_cred	date	Fecha en que la credencial de socio fue expedida
fecha_vencimiento_cred	date	Fecha de vencimiento de la credencial
vigente	int	Señala si la credencial de socio está vigente

Tabla 2

RENTA		
Campo	**Tipo**	**Descripción**
id_renta	int	El identificador único de una determinada renta efectuada
folio_socio	int	El socio al que se concedió la renta
id_pelicula	varchar	El identificador único de la película rentada
fecha_renta	date	Fecha en que efectuó la renta
fecha_devolucion	date	Fecha límite para entregar la película rentada sin generar cargos
fecha_real_devol	date	Fecha real en que se realizó la devolución de la película
activa	int	Indica si la renta está activa o ya está procesada

Tabla 3

ENCARGADO		
Campo	**Tipo**	**Descripción**
usuario	varchar	El identificador o nombre del usuario de la aplicación (encargado)
password	varchar	El password o clave de acceso a la palicación

Tabla 4

ESTADO		
Campo	**Tipo**	**Descripción**
id_estado	int	Es el identificador único de un estado de la república
estado	varchar	Es el nombre de un estado

Tabla 5

PELÍCULA		
Campo	**Tipo**	**Descripción**
id_pelicula	varchar	Es el identificador único de una película,
titulo	varchar	El título de la película
id_genero	int	El género al que pertenece la película
formato	varchar	El formato en que está disponible la película
precio_renta	double	El precio al que se renta la película
clasificación	int	La clasificación a la que pertenece la película

Tabla 6

CLASIFICACIÓN		
Campo	**Tipo**	**Descripción**
id_clasificacion	int	El identificador único de un tipo de clasificación de películas
clasificacion	varchar	La descripción de un tipo de clasificación

Tabla 7

GÉNERO		
Campo	**Tipo**	**Descripción**
id_genero	int	Es el identificador único de un tipo de género de películas
genero	varchar	La descripción del tipo de género

Tabla 8

CARGO		
Campo	**Tipo**	**Descripción**
id_renta	int	Es el identificador de la renta a la que pertenece el cargo
monto	double	Es la cantidad en dinero, del cargo por retraso en devolución
pagado	int	Indica si el cargo ya ha sido cubierto o no

ÁRBOL DE NAVEGACIÓN DEL SISTEMA

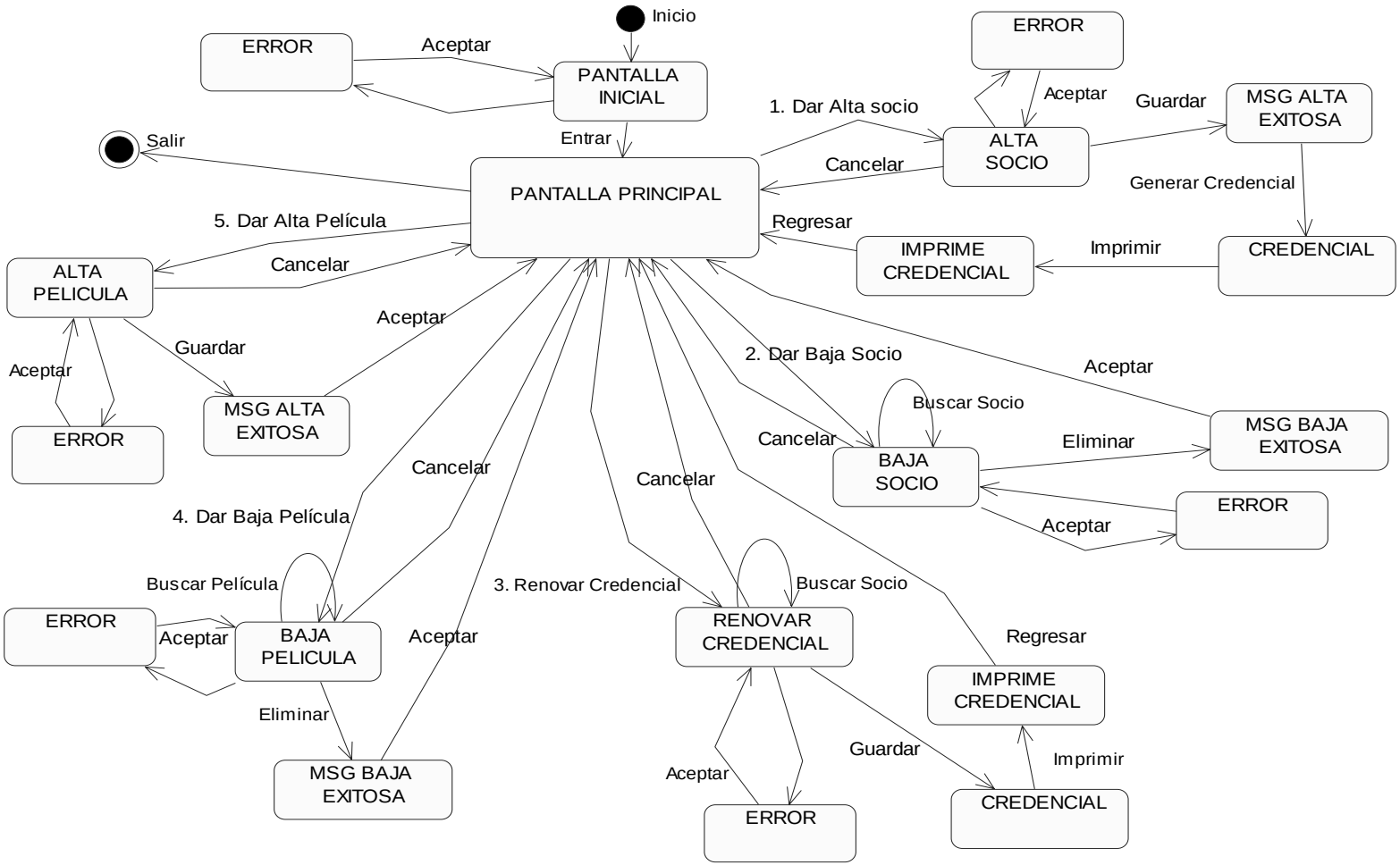


Figura 5.2. Mapa de Navegación o Diagrama de Estados parte I.

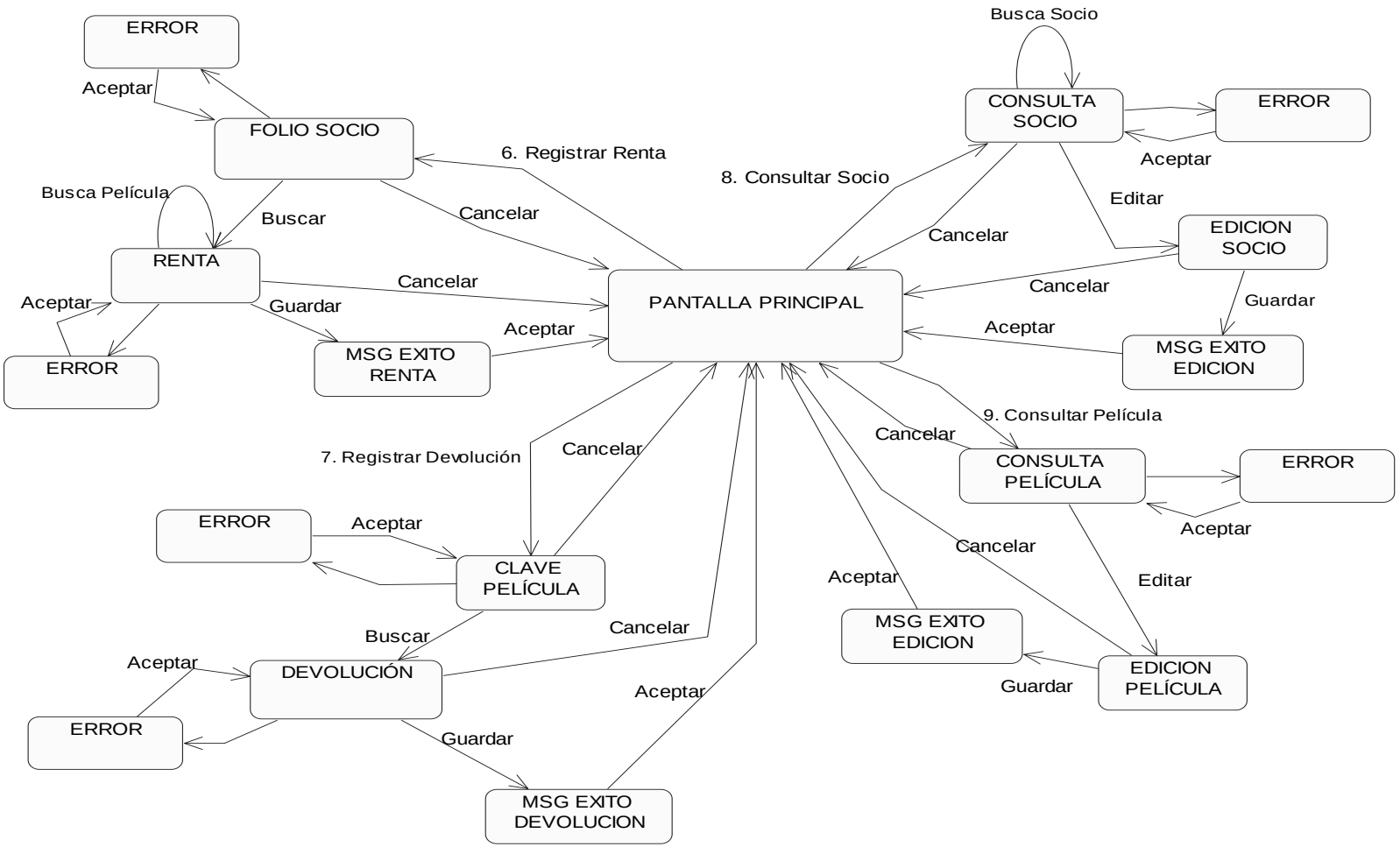


Figura 5.3. Mapa de Navegación o Diagrama de Estados parte II.

CODIFICACIÓN DE LAS CLASES MÁS IMPORTANTES

Como ya se menciono antes, para una clara separación de las reglas del negocio, el modelo y la capa de presentación, se utilizó una capa intermedia de Servicio, que solo sirve de enlace entre el controlador y los DAO. Los DAOs constituyen nuestros accesos, y consultas a la base de datos. La aplicación basa su funcionamiento en el siguiente diagrama. Donde tenemos una clase Action, que se comunica con una clase Servicio, la cual accede al una clase DAO; la clase DAO realiza los procesos solicictados en la base de datos y regresa los resultados a la clase Servicio, que a su vez regresa la respuesta al Action, quien decide que se debe hacer con dichos resultados. Para hacer la manipulación y transferencia de los datos se utilizan llas clases que representan los JavaBeans de negocio.

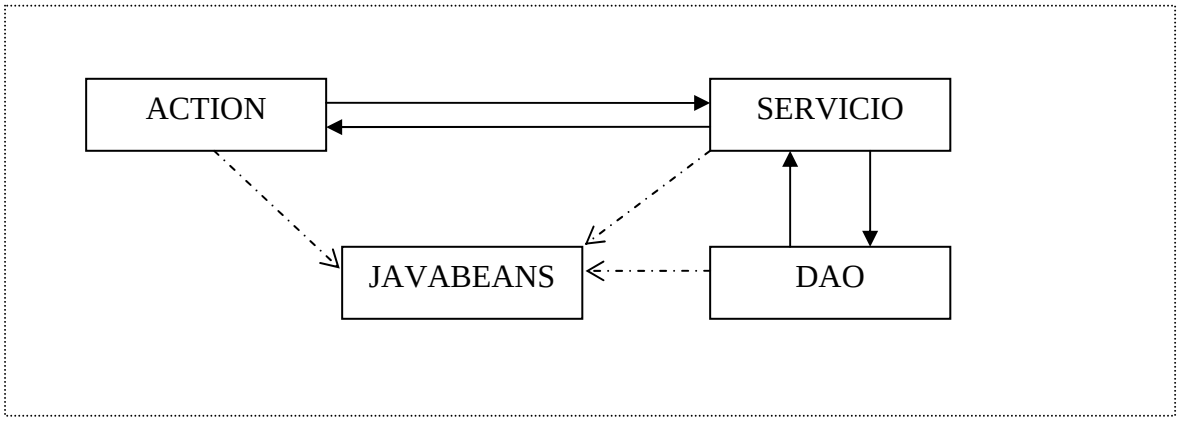


Figura 5.2. Esquema funcional del Sistema Videoclub.

Ahora mostramos el código final de las principales clases java de la aplicación, para aclarar un poco más los conceptos antes vertidos.

LoginAction

```
/*
 *Clase Action que efectúa la autentificación del encargado para permitir o denegar el acceso.
 * LoginAction.java
 * @author: M.Y.P.
 */
package com.diplomado.videoclub.Actions;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForward;
import com.diplomado.videoclub.Forms.*;
import com.diplomado.videoclub.VO.*;
import com.diplomado.videoclub.Service.*;

public class LoginAction extends Action {

   public ActionForward execute(ActionMapping mapping, ActionForm  form,
         HttpServletRequest request, HttpServletResponse response)
         throws Exception {


         //Creamos una instancia del Bean de Presentacion y recibimos los parámetros
           del formulario
        LoginForm UsuarioBean= (LoginForm) form;

         //Creamos una instancia del Bean de Negocio
        LoginVO bean = new LoginVO();

         //Guardamos los datos del Bean de Presentacion en el Bean de Negocio
        bean.setUsuario (UsuarioBean.getTxtUsuario ());
        bean.setPassword (UsuarioBean.getTxtPassword ());


          //Se crea una instancia del Servicio
        LoginService service = new LoginService();

          //Se llama al método que hace la validacion del usuario
        int i = service.validaUsu (bean);
        if(i==1){
            return (mapping.findForward("Acceso_Encargado"));
         } else{
            return (mapping.findForward("Acceso_Denegado"));
         }
    }
}
```

LoginForm

```
/*
 * Clase Form que sie utiliza para la transferencia de  los atributos del formulario al  Action que los manipula..
 * LoginForm.java
 * @author M.Y.P.
 */
package com.diplomado.videoclub.Forms;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;

public class LoginForm extends org.apache.struts.action.ActionForm {

   private String  txtUsuario;
   private String  txtPassword;

   public String getTxtUsuario () {
      return txtUsuario;
   }

   public void setTxtUsuario (String txtUsuario) {
      this.txtUsuario = txtUsuario;
   }

   public String getTxtPassword () {
      return txtPassword;
   }

   public void setTxtPassword (String txtPassword) {
      this.txtPassword = txtPassword;
   }
}
```

LoginVO

```
/*
 *Clase VO que sirve de contenedora de los atributos del formulario pero para uso en la capa de  lógica de negocios.
 *LoginVO.java
*/
package com.diplomado.videoclub.VO;
public class LoginVO {

   private String  usuario;
   private String  password;

   public String getUsuario() {
      return usuario;
   }
   public void setUsuario(String usuario) {
      this.usuario = usuario;
   } ..................
}
```

LoginService

```
/*
 * Clase Service que hace los procesos de la lógica de negocio accesando a los DAOs.
 * LoginService.java
* @author M. Y. P.
*/
package com.diplomado.videoclub.Service;
import com.diplomado.videoclub.DAO.LoginDAO;
import com.diplomado.videoclub.VO.LoginVO;

public class LoginService {
            public int validaUsu(LoginVO bean) throws Exception{
               int i;
               LoginDAO dao = new LoginDAO();
               i=dao.buscaUsu(bean);
             return i;
            }
}
```

LoginDAO

```
/*
 *Clase DAO que permite los accesos a la base de datos y  que hereda de Conexionado para obtener la conexión.
 * LoginDAO.java
 * @author M.Y.P.
*/
package com.diplomado.videoclub.DAO;
import java.sql.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import com.diplomado.videoclub.VO.*;

public class LoginDAO extends ConexionDAO{

  private static final String query= "Select usuario, password from encargado where usuario=? and password=?";
  public int buscaUsu(LoginVO bean) throws Exception{
    Connection cn = null;
    PreparedStatement  pstmt = null;
    ResultSet rs = null;
    int i=0;
    try {
      cn = this.getConnetion();
      pstmt = cn.prepareStatement(query);
      pstmt.setString(1, bean.getUsuario());
      pstmt.setString(2, bean.getPassword());
      rs = pstmt.executeQuery();
      while(rs.next()) {
        if (rs.getString(1).equals(bean.getUsuario()) && (rs.getString(2).equals(bean.getPassword()))) {
          i = 1;
        }
      }
    } catch (Exception e){
      e.printStackTrace();
    } finally {
      if (rs != null){   rs.close();  }
```

```
        if (pstmt != null){ pstmt.close();  }
        if (cn != null){   cn.close();  }
      }
      return i;
   }
}
```

ConexionDAO

```
/*
 * Clase DAO utilizada para establecer la conexión a través del pool de conexiones definido en  context.xml.
 * Además es utilizada para albergar métodos de uso general para las los DAOs.
 * ConexionDAO.java
 * @author M.Y.P.
*/
package com.diplomado.videoclub.DAO;
import java.sql.Connection;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import java.text.SimpleDateFormat;
import java.util.Date;

public class ConexionDAO {

 private Connection cn;

   public Connection getConnetion () {
      try {
         Context init = new InitialContext();
         DataSource ds = (DataSource) init.lookup("java:comp/env/jdbc/videoclub");
         cn = ds.getConnection();
      } catch (Exception e){
         cn = null;
      }
      return cn;
   }


   public String obtieneFecha() {

      Date fecha = new Date();
      SimpleDateFormat formato = new SimpleDateFormat("yyMMdd");
      String cadenafecha = formato.format(fecha);
      return cadenafecha;
    }
 }
```

Así como en el caso de la actividad de autentificación del usuario, se requirió un Action, un Form, un Service, Un VO y un DAO, de la misma forma se procede para todos los casos de uso restantes. Por lo cual, a continuación sólo se presenta el Action y el DAO del caso de uso: Consultar Socio y al final se muestran la pantalla que genera.

ConsultaSocioAction

```
/*
 * Clase Action que se ejecuta cuando se solicita el proceso de consultar socio.
 * ConsultaSocioAction.java
 * @author M.Y.P.
 */
package com.diplomado.videoclub.Actions;
import com.diplomado.videoclub.Forms. ConsultaSocioForm;
import com.diplomado.videoclub.Service.SocioService;
import com.diplomado.videoclub.VO.SocioVO;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForward;
import org.apache.struts.actions.DispatchAction;

public class ConsultaSocioAction extends DispatchAction {

  public ActionForward trySocio(ActionMapping mapping, ActionForm  form,
      HttpServletRequest request, HttpServletResponse response)
      throws Exception {

      ConsultaSocioForm forma = (ConsultaSocioForm)form;
      SocioVO bean = new SocioVO();
      SocioVO bean1 = new SocioVO();
      bean.setSocio(forma.getTxtSocio());
      SocioService service = new SocioService();
      bean1 = service.traerSocio(bean);

      forma.setTxtNombre(bean1.getNombre());
      forma.setTxtApePat(bean1.getApePat());
      forma.setTxtApeMat(bean1.getApeMat());
      forma.setTxtIfe(bean1.getIfe());
      forma.setTxtCalle(bean1.getCalle());
      forma.setTxtColonia(bean1.getColonia());
      forma.setTxtEstado(bean1.getEstate());
      forma.setTxtTel(bean1.getTel());
      forma.setTxtCorreo(bean1.getCorreo());

      request.setAttribute("EditaSocioForm", forma);
      request.getSession().setAttribute("Soc", bean);
    return mapping.findForward("trae");
  }

   ..............................
}
```

SocioDAO

```
/*
 *Clase DAO que contiene todos los accesos a la base para procesos  relacionados con los socios.
 * SocioDAO.java
 * @author M.Y.P.
 */
package com.diplomado.videoclub.DAO;
import com.diplomado.videoclub.VO.EdoVO;
import com.diplomado.videoclub.VO.SocioVO;
import java.sql.*;
import java.util.*;
public class SocioDAO extends ConexionDAO{
   Connection cn = null;
   PreparedStatement  pstmt = null;
   ResultSet rs = null;
private static final String query2 = "Select nombre, ap_paterno, ap_materno, clave_ife, calle, colonia, id_estado, telefono, correo,
fecha_alta from socio where folio_socio=?";
 ..............................
 ..............................
   public SocioVO buscaSocio(SocioVO bean) throws Exception{
      Connection cn = null;
      PreparedStatement  pstmt = null;
      ResultSet rs = null;
      SocioVO bean1= new SocioVO();
      try {
         cn = this.getConnetion();
         pstmt = cn.prepareStatement(query2);
         pstmt.setInt(1, bean.getSocio());
         rs = pstmt.executeQuery();
         while(rs.next()) {
            bean1.setFecha(rs.getString(10));
            bean1.setNombre(rs.getString(1));
            bean1.setApePat(rs.getString(2));
            bean1.setApeMat(rs.getString(3));
            bean1.setIfe(rs.getString(4));
            bean1.setCalle(rs.getString(5));
            bean1.setColonia(rs.getString(6));
            bean1.setEstado(rs.getInt(7));
            bean1.setTel(rs.getString(8));
            bean1.setCorreo(rs.getString(9));
         }

      } catch (Exception e){
         e.printStackTrace();
      } finally {  if (rs != null){    rs.close(); }
         if (pstmt != null){  pstmt.close();  }
         if (cn != null){  cn.close();     }
      }
      return bean1;
   }
}
```

El resultado de la ejecución de las clases anteriores delproceso “Consulta Socio” genera la siguiente pantalla. Donde el proceso es así; primero el usuario escribe el folio del socio en la casilla correspondiente y pulsa el botón buscar, el sistema busca los datos de ese socio en la base y llena los campos correspondientes en la misma pantalla con los resultados:

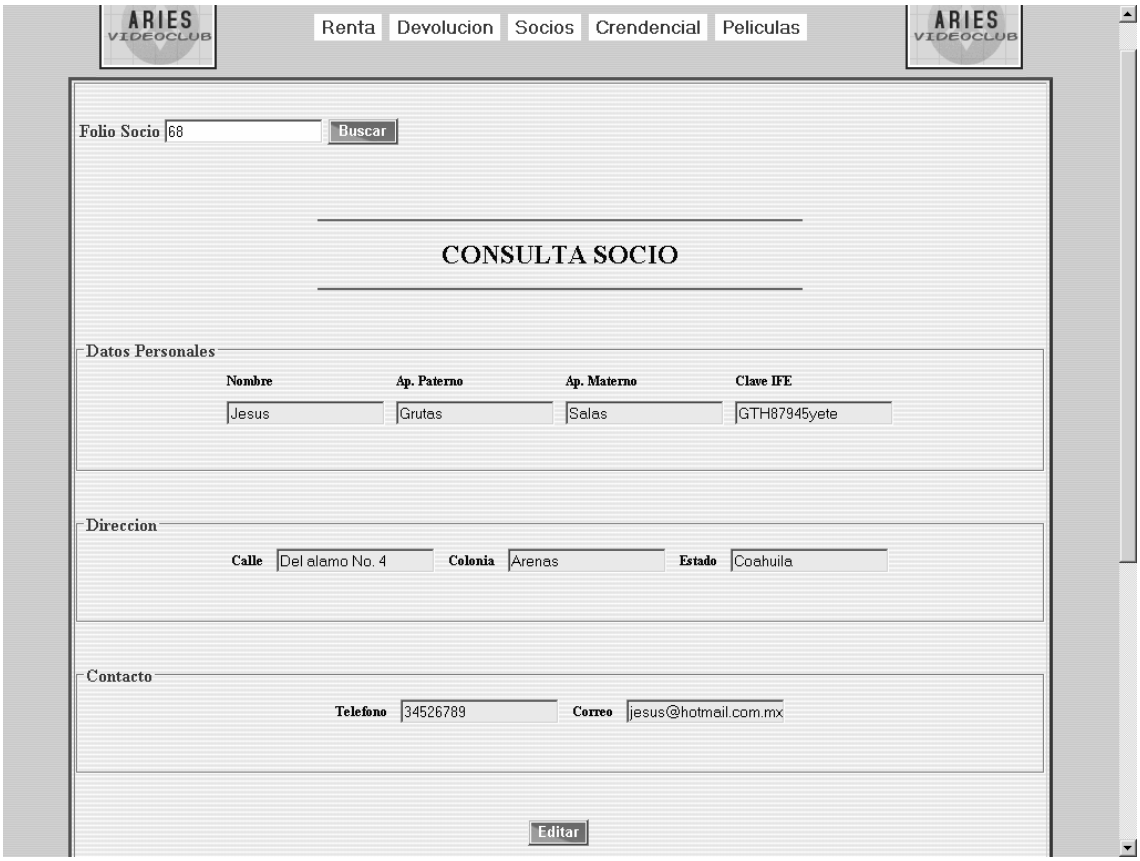


Figura 5.3. Pantalla de consulta de Socio después de procesar la búsqueda.

Aquí mostramos el código de la página JSP correspondiente.

consulta_socio.jsp

```
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>

<html:html>

   <head>
      <script src="./JS/javascript.js" type="text/javascript"></script>
      <link rel="stylesheet" href="CSS/estilo.css" type="text/css">
   </head>

   <html:form  action="editaSocio.do?method=trySocio" styleId="frmBaja">
```

```
<table>
    <tr>
        <td class="Estilo2">
            Folio Socio
        </td>
        <td>
            <html:text property="txtSocio"/>
        </td>
        <td>
            <html:submit value="Buscar" styleClass="boton"/>
        </td>
    </tr>
</table>
<br>
<br>
<br>
<hr class="linea"/>
<table border="0" align="center" cellpadding="5" cellspacing="5">
    <tr>
        <td class="Estilo1" class="Estilo2">
            CONSULTA SOCIO
        </td>
    </tr>
</table>
 <hr class="linea"/>
<br>
<br>
<fieldset><legend class="Estilo2">Datos Personales </legend>
    <table align="center" cellpadding="" cellspacing="10">
        <tr>
            <td class="Estilo3">
                Nombre
            </td>
            <td class="Estilo3">
                Ap. Paterno
            </td>
            <td class="Estilo3">
                Ap. Materno
            </td>
            <td class="Estilo3">
                Clave IFE
            </td>

        </tr>
        <tr>
            <td>
                <html:text property="txtNombre" styleClass="deshabilita" readonly="true"/>
            </td>
            <td>
                <html:text property="txtApePat" styleClass="deshabilita" readonly="true"/>
            </td>
            <td align="left">
                <html:text property="txtApeMat" styleClass="deshabilita" readonly="true"/>
            </td>
            <td class="Estilo3">
                <html:text property="txtIfe" styleClass="deshabilita" readonly="true"/>
```

```
            </td>
        </tr>
        <tr>
            <td>
                 
            </td>
        </tr>
    </table>
</fieldset>
<br>
<br>
<fieldset><legend class="Estilo2">Direccion </legend>
    <table align="center" cellspacing="10">
        <tr>
            <td class="Estilo3">
                Calle
            </td>
            <td>
                <html:text property="txtCalle" styleClass="deshabilita" readonly="true"/>
            </td>
            <td class="Estilo3">
                Colonia
            </td>
            <td>
                <html:text property="txtColonia" styleClass="deshabilita" readonly="true"/>
            </td>
            <td class="Estilo3">
                Estado
            </td>
            <td align="left">
                <html:text property="txtEstado" styleClass="deshabilita" readonly="true"/>
            </td>

        </tr>
        <tr>
            <td>
                 
            </td>
        </tr>
    </table>
</fieldset>
<br>
<br>
<fieldset><legend class="Estilo2">Contacto</legend>
    <table align="center" cellspacing="10">
        <tr>
            <td class="Estilo3">
                Telefono
            </td>
            <td class="Estilo3">
                <html:text property="txtTel" styleClass="deshabilita" readonly="true"/>
            </td>
            <td class="Estilo3">
                Correo
            </td>
            <td class="Estilo2" align="left">
```

```
                <html:text property="txtCorreo" styleClass="deshabilita" readonly="true"/>
            </td>
        </tr>

        <tr>
            <td>
                 
            </td>
        </tr>
    </table>
</fieldset>
<br>
<br>
<table align="center">
    <tr>
        <td>
            <html:button property="borra" value="Editar" styleClass="boton" onclick="editar();"/>
        </td>
        <td>
            <!-- property="borra" value="Guardar" styleClass="boton" disabled="true"/>-->
        </td>
    </tr>
</table>

</html:form>


</html:html>
```

PANTALLAS PRINCIPALES

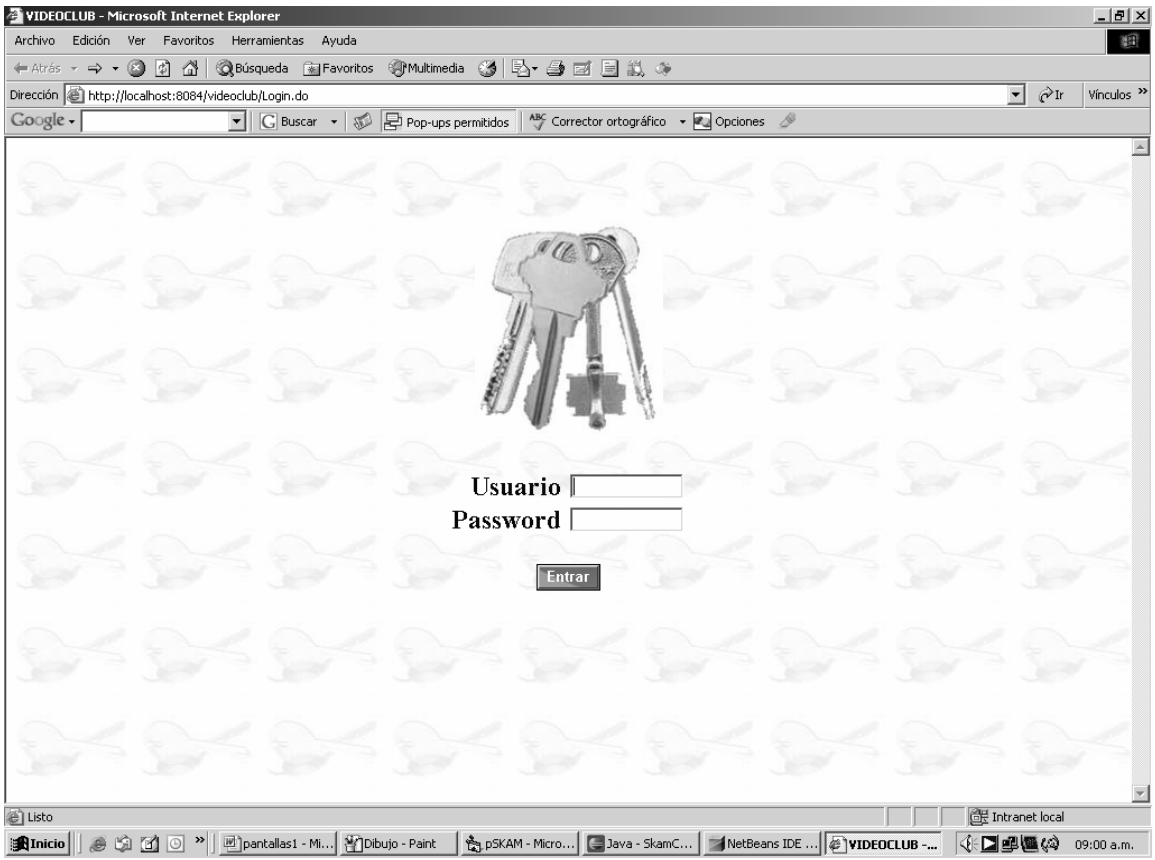


Figura 5.4. Pantalla de inicio de la aplicación, donde se capturan el usuario y el password para poder acceder a la pantalla principal del sistema.

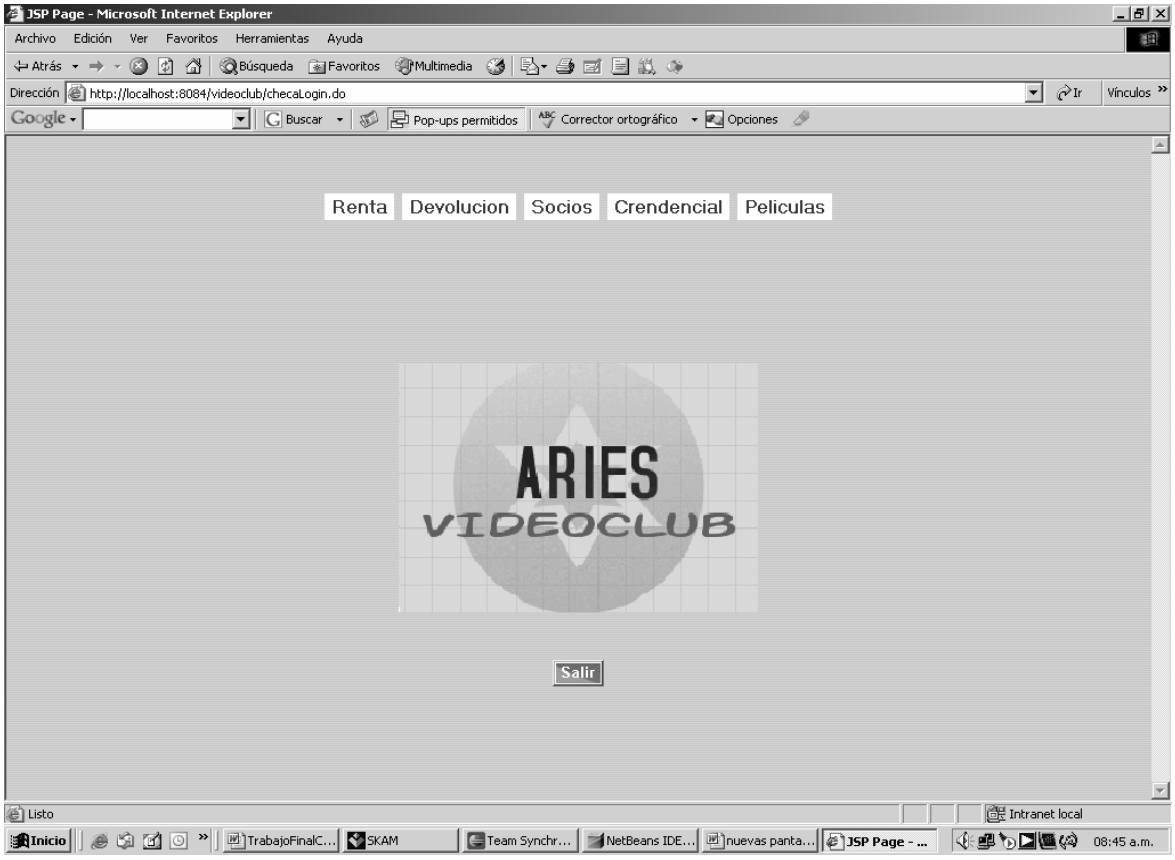


Figura 5.5. Pantalla principal de la aplicación, cuando ya se ha permitido el acceso al usuario, donde se presenta un menú desplegable con las posibles actividades a realizar, y un botón para salir de la aplicación.

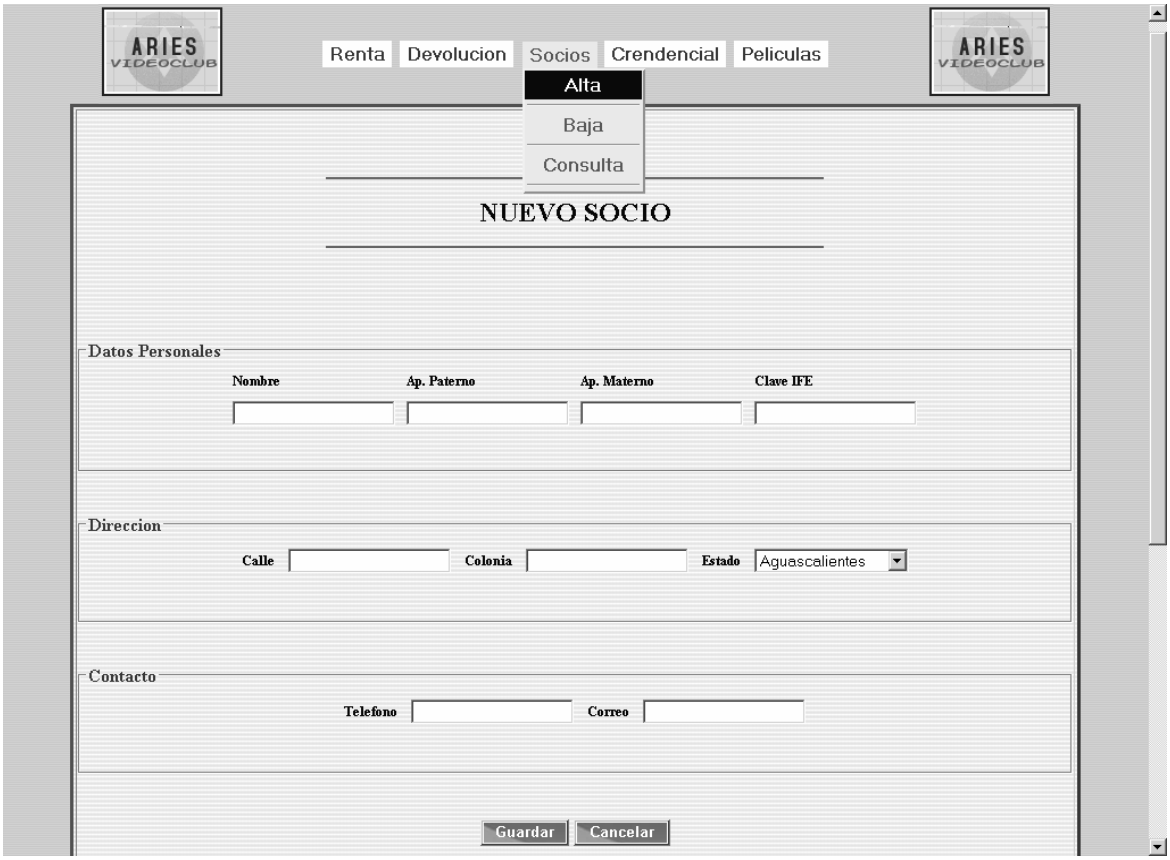


Figura 5.6. Pantalla inicial al elegir la actividad de “Alta de Socio”. En donde se permite capturar en tres apartados diferentes los datos del nuevo socio. Primero, los datos personales, después su dirección y por último telefono y correo. Se presenta un botón para guardar los datos del socio y otro para cancelar la operación.

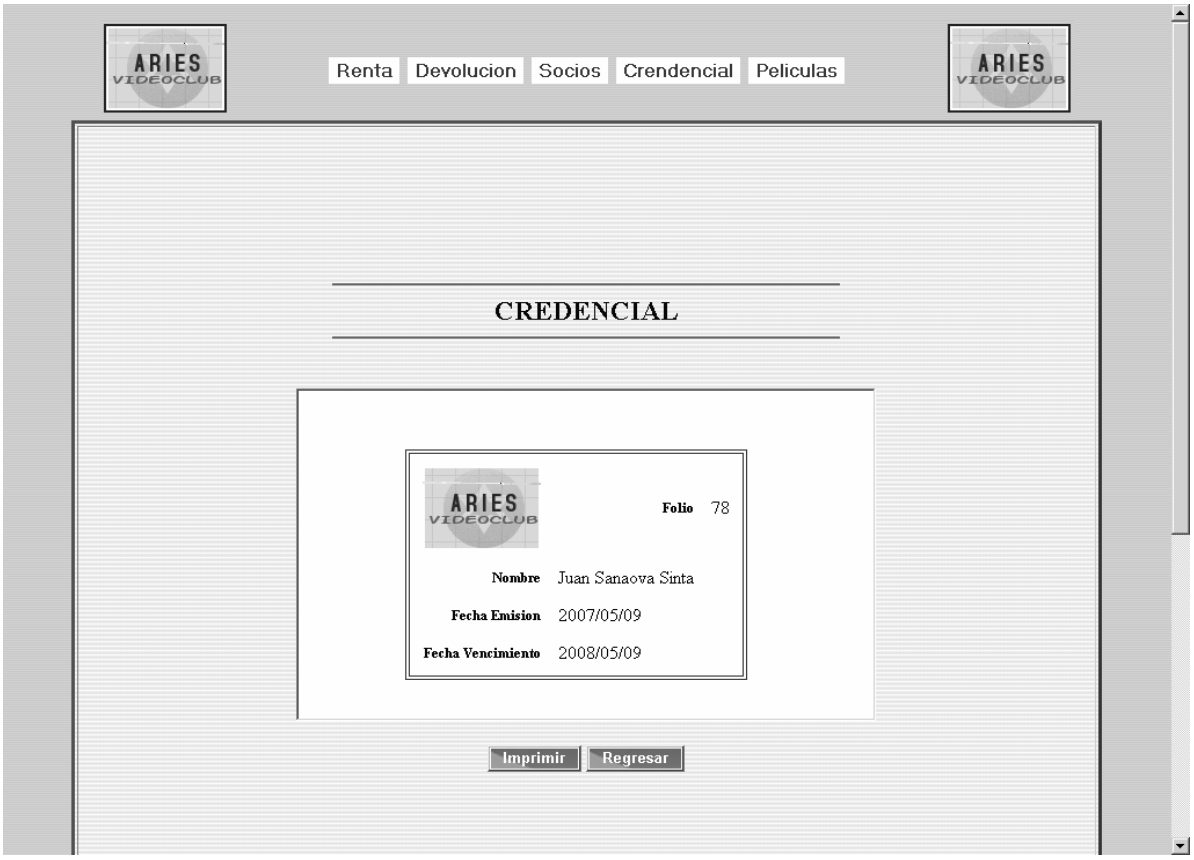


Figura 5.7. Pantalla quecontiene los datos de la credencial que genera el sistema, cuando el socio ha sido de alta exitosamente, dando la opción al usuario de imprimir la credencial o de regresar.

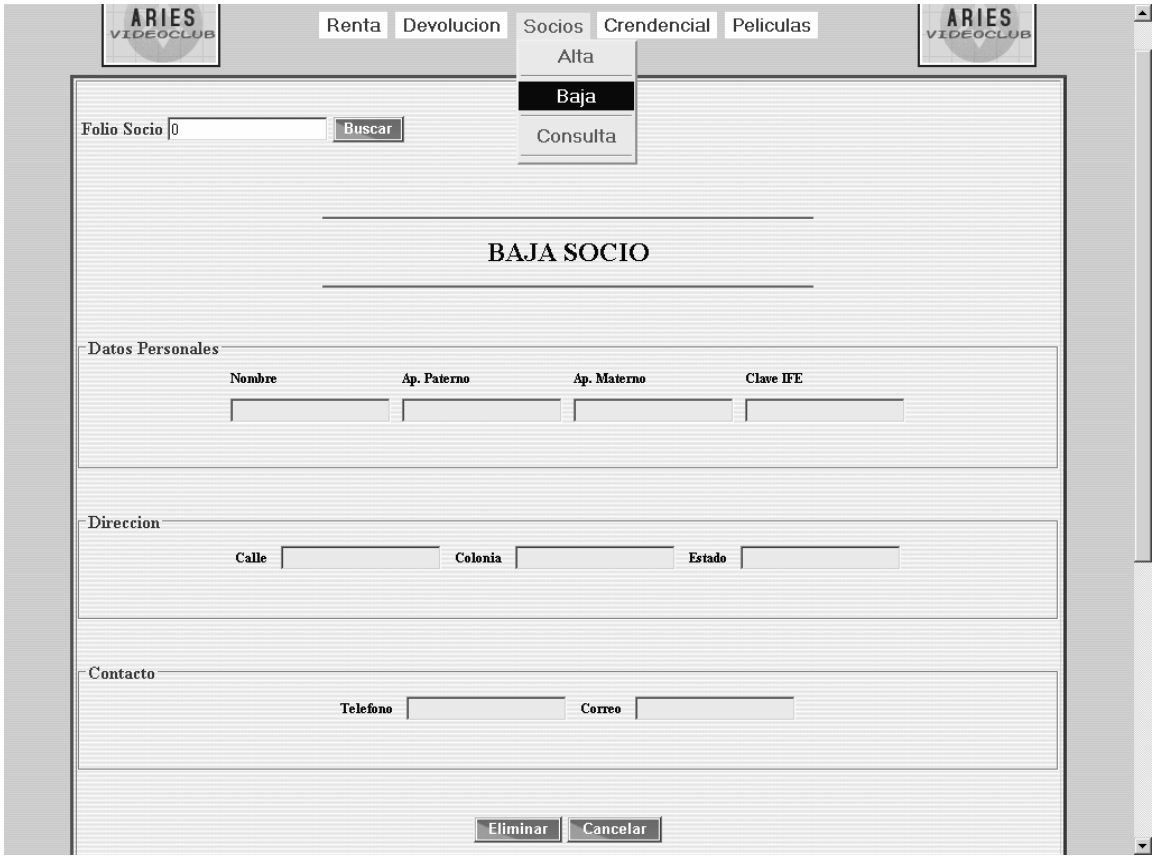


Figura 5.8. Pantalla principal que se genera cuando es elegido el servicio para dar de baja a un socio. El usuario captura el folio del socio que desea eliminar, pulsa buscar y los datos de ese socio son cargados en los campos correspondientes de la misma pantalla. Se tiene un botón para eliminar al socio y otro para cancelar.

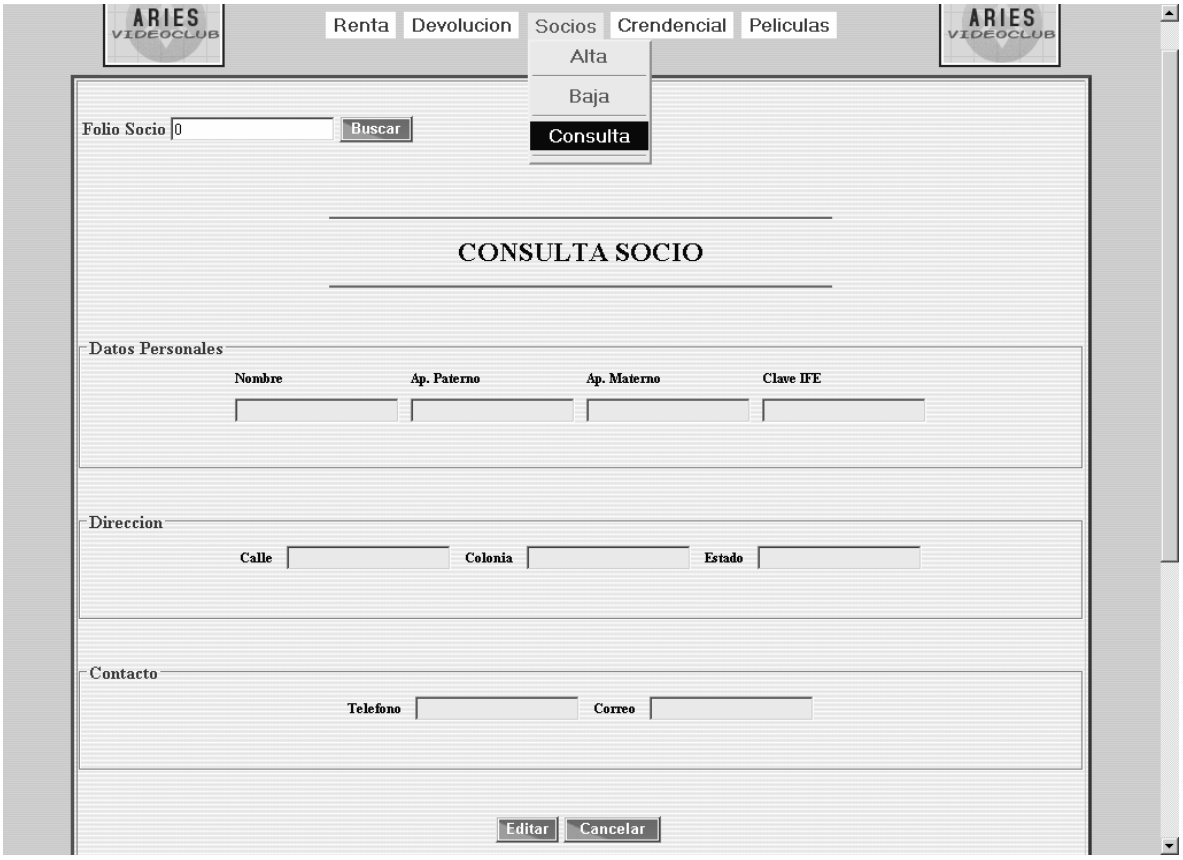


Figura 5.9. Pantalla principal para consultar los datos de un socio. Tiene la misma funcionalidad que el servicio dar de baja (anterior). Además en esta pantalla se tiene el botón para editar los datos de un socio en específico y un botón para cancelar.

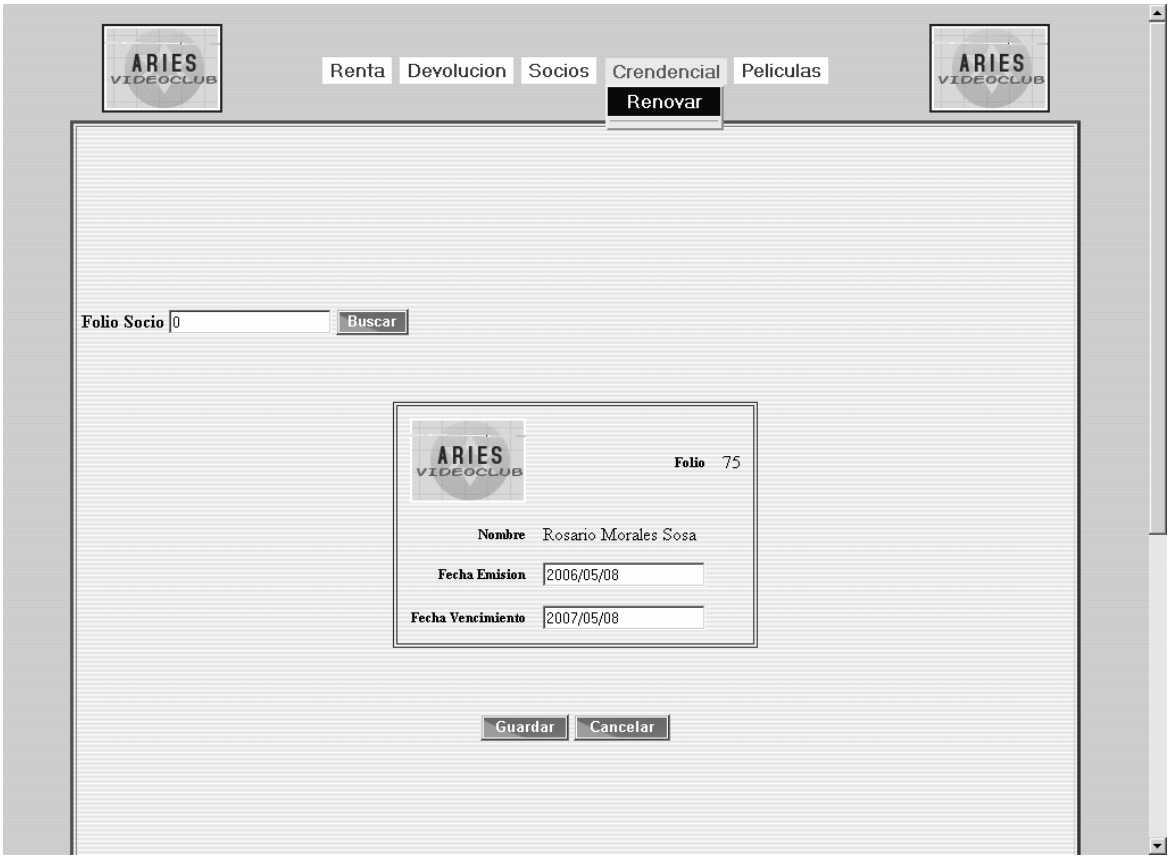


Figura 5.10. Pantalla principal cuando se desea renovar una credencial de socio por otro año más de vigencia. Aquí el usuario captura el folio del socio, pulsa el botón de buscar y en el formato de credencial que aparece en la parte inferior de la pantalla se cargan los datos de la credencial de ese socio, se encuentran habilitados los campos de la fecha de emisión y fecha de vencimiento para que el usuario pueda modificarlos y después pulsar el botón guardar o el de cancelar.

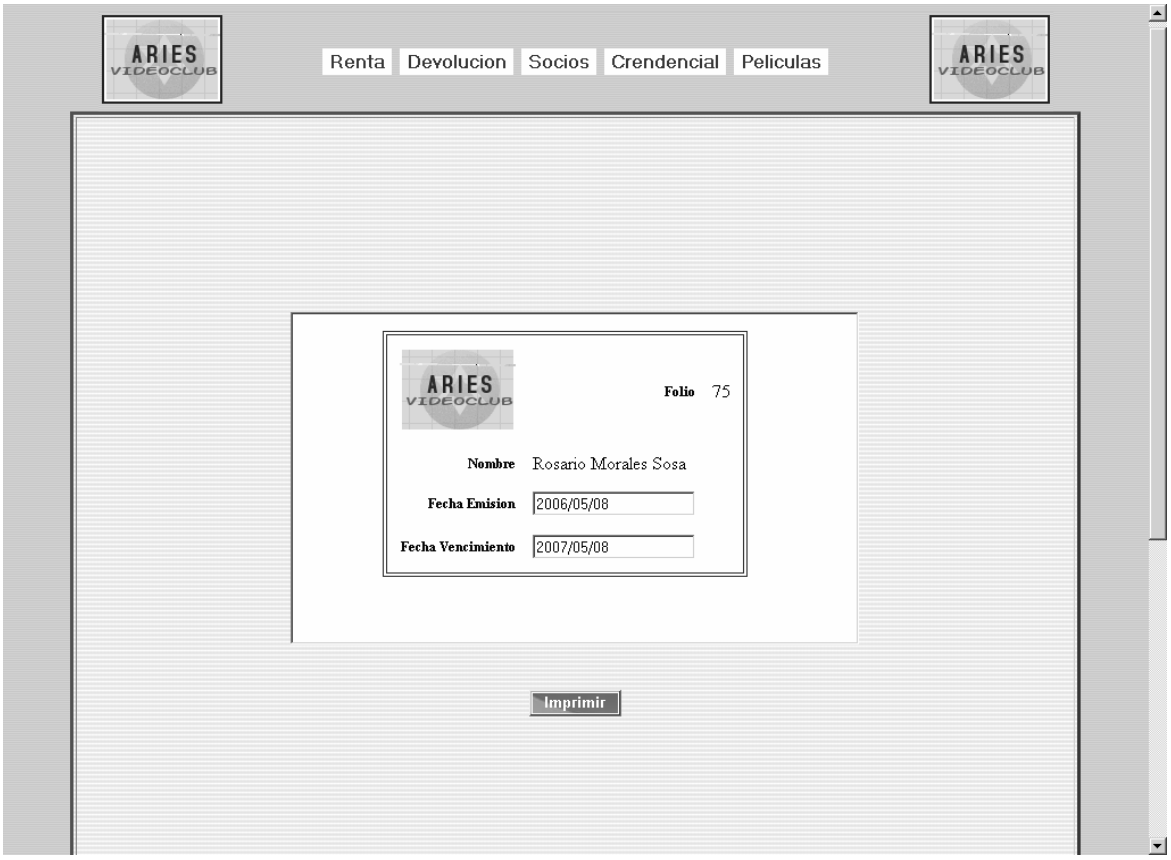


Figura 5.11. Pantalla que se genera después de haber modificado la vigencia de la credencial de socio (en la pantalla anterior), para permitir imprimir la nueva credencial generada.

Figura 5.12. Pantalla principal cuando se activa el servicio de agregar una película al catálogo. En ella, se capturan los datos más importantes de la película que se quiere dar de alta, posteriormente se pulsa el botón guardar o en su defecto el de cancelar.

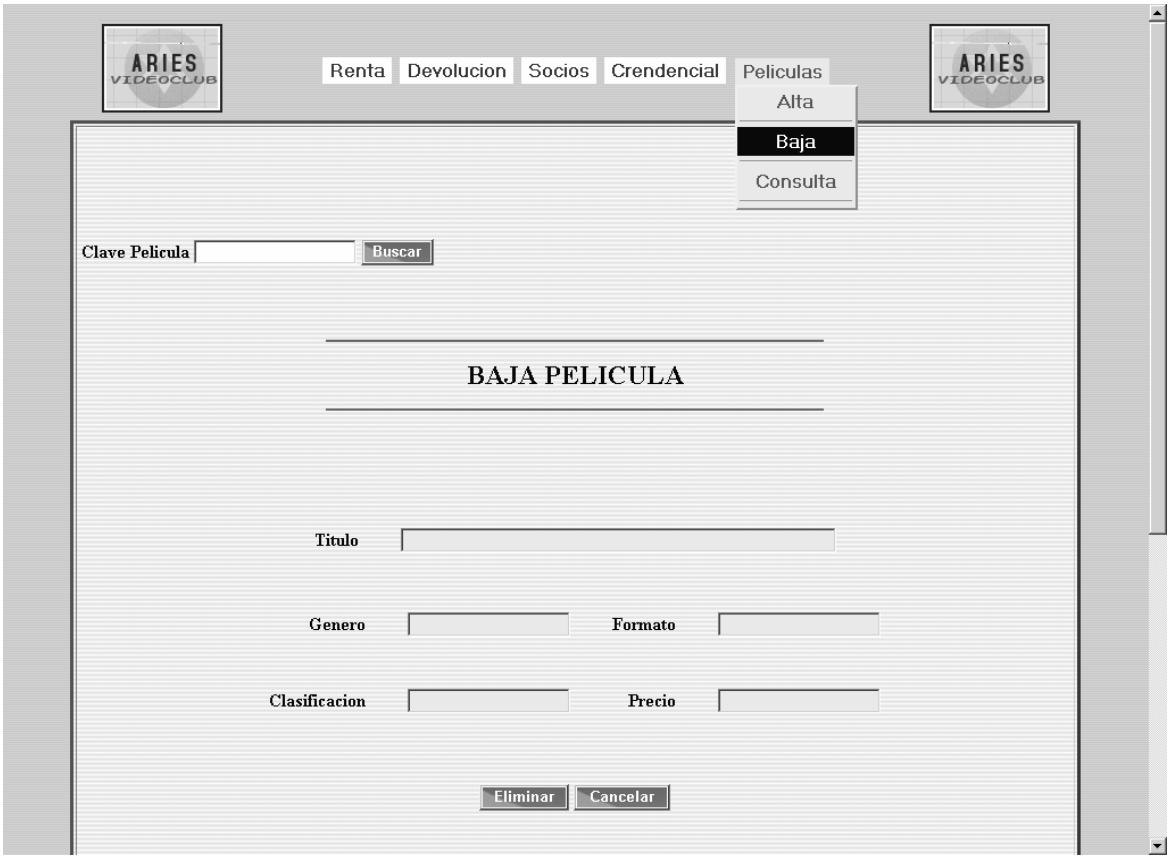


Figura 5.13. Pantalla que se genera al elegir la actividad para dar de baja una película en el menú superior de la pantalla. El funcionamiento es el siguiente, el usuario captura la clave de la película, pulsa el botón buscar y los datos de esa película se cargan en los campos correspondientes de la parte inferior de la pantalla. Después se pulsa el botón eliminar o el botón para cancelar.

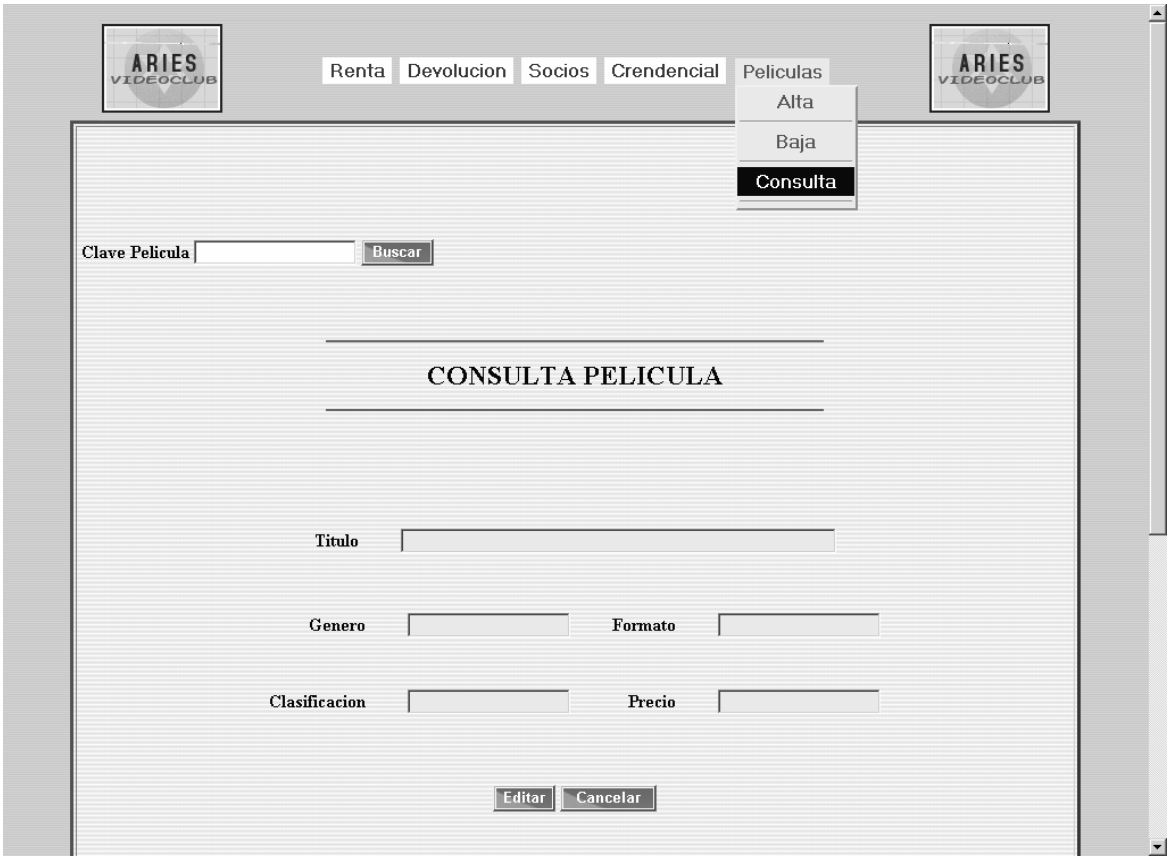


Figura 5.14. Cuando se elige consultar los datos de una película, se abre ésta pantalla, donde se captura la clave de la película y se pulsa el botón buscar. Entonces el sistema busca los datos de esa película y los carga en los campos de la parte inferior de la pantalla. Aquí el usuario puede pulsar editar para efectuar cambios o pulsar cancelar.

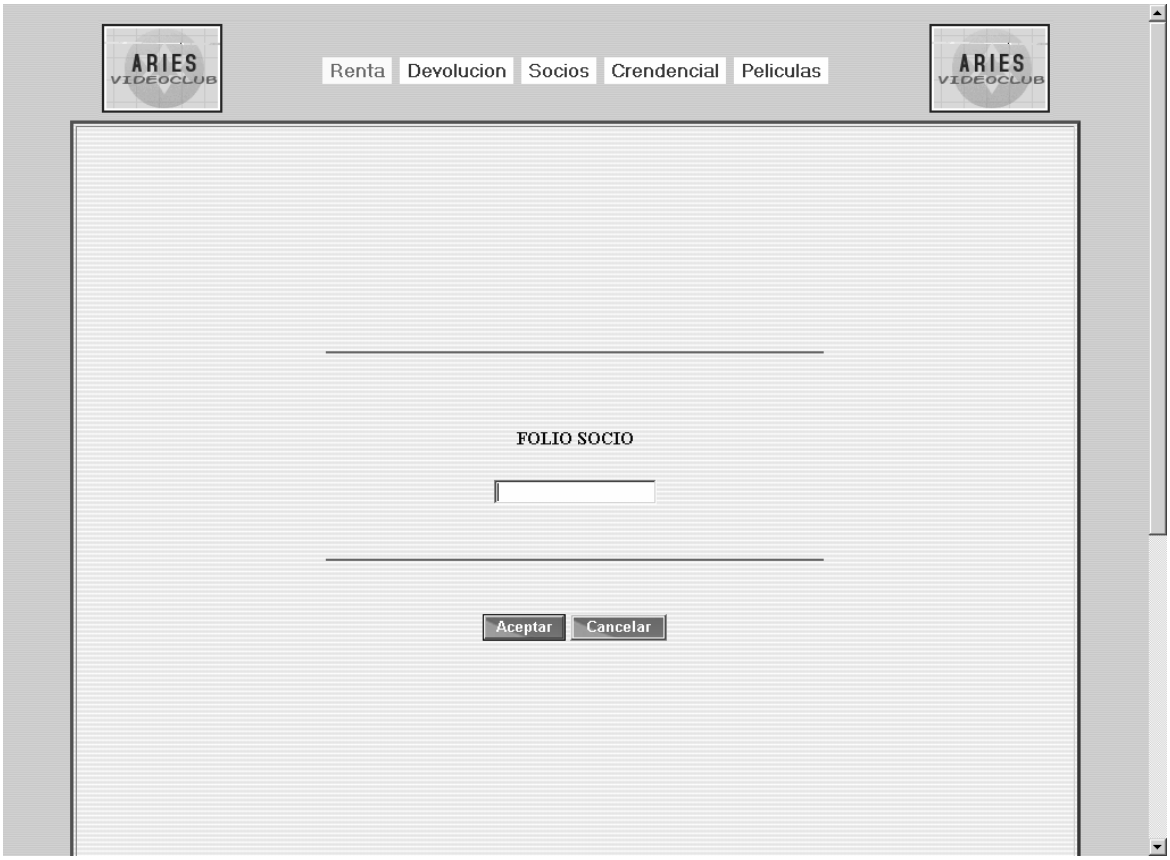


Figura 5.15. Cuando el usuario desea registrar una nueva renta, aparece ésta pantalla, en donde se permite capturar el folio del socio, se tiene el botón aceptar para que el sistema inicie la búsqueda de los datos del socio y también la opción para cancelar.

Figura 5.16. A continuación de que el usuario pulsa el botón aceptar de la pantallaanterior, se genera ésta pantalla que carga los datos del socio en la parte superior. Después aparece el campo donde se captura cada una de las claves de las películas que se pretende rentar, las cuales irán apareciendo en la parte inferior. Cuando se terminen de capturar todas las películas, se podrá pulsar el botón guardar para almacenar los datos de la renta o en su defecto el botón cancelar para abortar la operación.

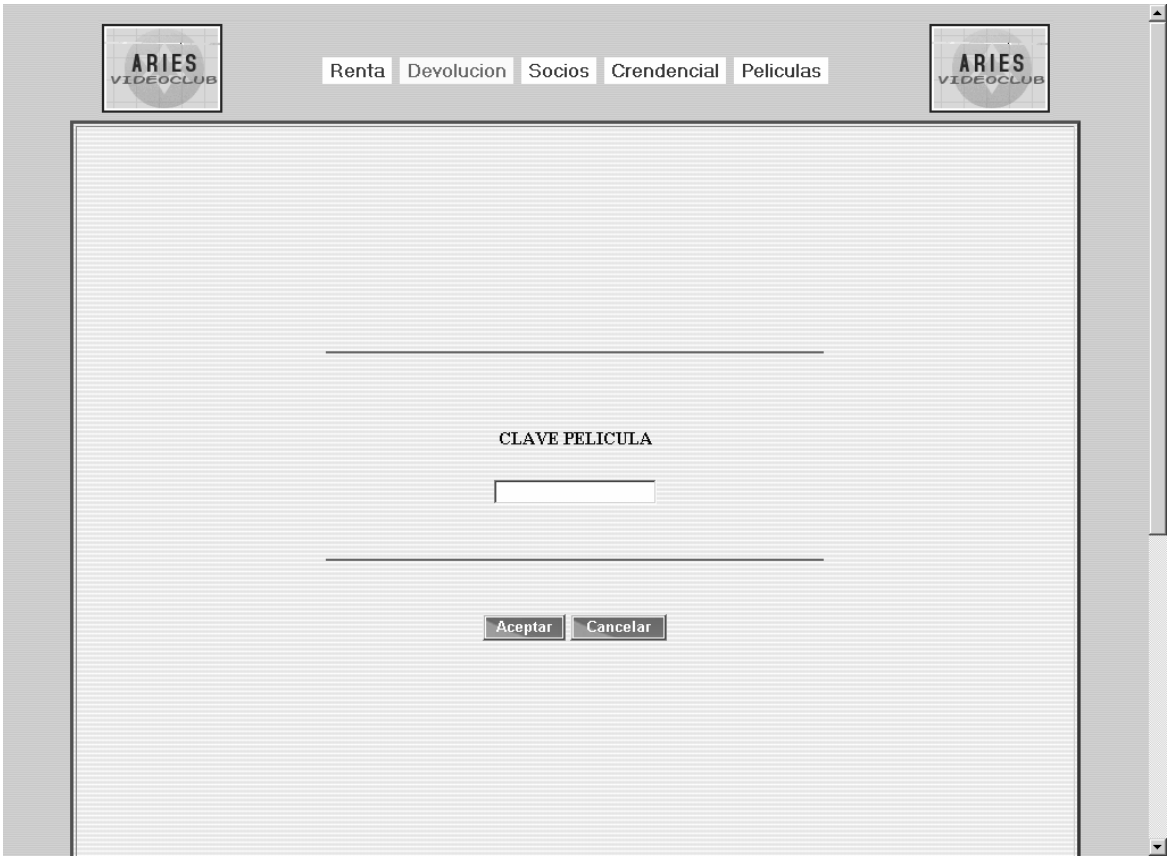


Figura 5.17. Pantalla principal para el registro de la devolución de una película. Se permite la captura de la clave de la película en cuestión y se podrá pulsar el botón aceptar para que el sistema inicie la búsqueda del registro de renta que incluye a esa película, o pulsar el botón para cancelar la operación.

Figura 5.18. Cuando en la pantalla anterior se pulsa el botón aceptar, el sistema genera ésta pantalla, donde aparecen los datos de la película (si existen en la base) que se está devolviendo. También existe la posibilidad de abortar la operación, pulsando el botón cancelar.

EVALUACIÓN DEL SISTEMA

Aunque por falta de espacio, no se agregaron todas las clases que controlan la aplicación, y todas las pantallas que el sistema usa, con las que se incluyeron es más que suficiente para dar una clara perspectiva de lo que es capaz de realizar; por lo que se pueda establecer que es funcional, eficiente y fácil de usar.

El sistema web desarrollado a lo largo de este trabajo de investigación nos ha demostrado la utilidad del framework Struts para las aplicaciones web. No es que sea la panacea del mundo, pero si es una herramienta, digamoslo así, para realizar sistemas web de manera eficiente y a nivel profesional.

El haber utilizado dicho framework como base para nuestro proyecto, nos ha facilitado en gran manera la organización, estructuración y delimitación de la aplicación. Además, aunque no está dentro del alcance del presente trabajo, pero Struts se puede complementar para trabajar armónicamente junto con otros frameworks que están orientados a otros aspectos del manejo de un sistema web; para poder manejar por ejemplo transacciones, seguridad, etc.

Durante todo el desarrollo de nuestra aplicación se han tratado de conjuntar y utilizar las herramientas más importantes que se encuentran disponibles para crear sistemas en el ambiente de Internet. Se ha tratado de demostrar que dichas herramientas se pueden utilizar de manera armónica, todos cumpliendo un propósito bien definido.

El sistema final cumple con los objetivos planteados, ya que el usuario quedó satisfecho con la implementación de los procesos que se siguen en el videoclub. El sistema es seguro, como parte central de este aspecto tenemos la funcionalidad eficiente de la validación de usuarios, así como el hecho de que la información mantiene todo el tiempo su integridad y congruencia.
Otro de los objetivos que se logró fue la clara separación del modelo, la vista y controladores. Además el hecho de haber implementado el Patrón de diseño MVC, permitió también una mayor velocidad y calidad en el proceso de desarrollo, debido a que fue posible implementar cada una de las capas del sistema (Modelo, Vista, Controlador) y hacer modificaciones sin afectar a las demás. Desde esta perspectiva, esto es de mucha utilidad en la creación de grandes sistemas porque alienta la distinción de perfiles de programador, es decir, un programador se puede dedicar a la capa de presentación, otro a la capa de negocio, etc., dejando la integración de todo el sistema para el momento en que todos estos componentes se hayan probado eficientemente al realizar sus procesos por separado.

También al usar el MVC el sistema se beneficia de su modularidad, un diseño claro y ampliamente aceptado, facilidad para crear distintas vistas del mismo modelo de negocio y su extensibilidad

Hay una clara separación entre los componentes de un programa. Hay un API muy bien definido, cualquiera que use el API podrá reemplazar el modelo, la vista o el controlador sin aparente dificultad.

La conexión entre el modelo y sus vistas es dinámica, se produce en tiempo de ejecución, no en tiempo de compilación.

Por su parte, la interfaz de la aplicación, fue relativamente fácil de estructurar gracias al manejo de los Tiles de Struts, solamente se requirió la creación del concepto que se manejaría como base en todas las pantallas, hacer las respectivas maquetas o esqueletos a ir cargando los elementos específicos, por lo mismo hubo un incremento en la rapidez de su desarrollo. La interfaz final es fácil de usar, intuitiva, sencilla y cumple con todas especificaciones que es usuario deseaba ver plasmadas en las pantallas para realizar eficientemente su trabajo. Además cumple con los requerimientos mínimos de usabilidad.

Usabilidad se define como el grado en el que un producto puede ser utilizado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un determinado contexto de uso. Por tanto, la usabilidad del sistema no es un atributo inherente al software, no puede especificarse independientemente del entorno de uso y de los usuarios concretos que vayan a utilizar el sistema.

La usabilidad de nuestro sistema, se descompone en los siguientes aspectos básicos:

- Facilidad de aprendizaje: Es relativamente fácil de aprender la funcionalidad básica del sistema, como para ser capaz de realizar correctamente las tareas que desea realizar el usuario.
- Eficiencia: Se ha logrado la máxima velocidad de realización de tareas del usuario. Cuanto mayor es la usabilidad de un sistema, más rápido es el usuario al utilizarlo, y el trabajo se realiza con mayor rapidez.
- Recuerdo en el tiempo: Los usuarios intermitentes (que no utilizan el sistema regularmente) son capaces de usar el sistema sin tener que aprender cómo funciona partiendo de cero cada vez. Este atributo refleja el recuerdo acerca de cómo funciona el sistema que mantiene el usuario, cuando vuelve a utilizarlo tras un periodo de no utilización.
- Satisfacción: Éste es el atributo más subjetivo. Muestra la impresión subjetiva que el usuario obtiene del sistema. En este caso se ha logrado un grado de satisfacción aceptable.

Para concluir es necesario establecer que el Sistema Web para el videoclub, está abierto a mejoras y nuevas implementaciones que eficienten aún más su funcionalidad. Con todo, las características mínimas para el control de un videoclub están consideradas. Aunque debe aclararse que el propósito de este tratado, como se planteó desde el principio no fue desarrollar un sistema web en su totalidad, sino demostrar los pasos a seguir para lograr crear una aplicación web desde cero.

CONCLUSIONES

Desarrollar un sistema en el ambiente web, es una actividad de gran importancia en el mundo de la creación de software.
No obstante, aún no se deben “echar las campanas al vuelo”, ya que si bien es cierto que la implantación de un sistema web implica muchas ventajas en cuanto a costo-beneficio, también es cierto que las aplicaciones que funcionan en ese ambiente presentan muchas deficiencias, sobre todo en cuanto a seguridad e integridad de la información, por el grado de exposición que implica la propia naturaleza de Internet.

A diario escuchamos de fraudes y robos electrónicos, sobre todo a instituciones bancarias y que se dedican al manejo de dinero específicamente. Ésto se debe a los huecos de seguridad que no se han solucionado.

Los gobiernos mundiales en cooperación con las grandes empresas vinculadas con las tecnologías de información, han hecho y hacen cada día cuantiosas inversiones para contrarrestar al mínimo dichos problemas, es digno de mención que en muchos de los casos se han logrado avances notorios.

Sin embargo, los expertos consideran que todavía faltan algunas décadas para que se pueda decir con toda certeza que un sistema web es totalmente seguro.

Por otra parte, la tecnología de internet es cada vez más amplia, y no sólo estamos hablando del hardware, es decir de las máquinas. También estamos hablando de su programación. Las opciones son cada vez más en número y en complejidad: javascript, vbscript, ASP, SQL, java, html, dhtml, xml, etc, sólo por mencionar las más populares.

El lenguaje de marcado tradicional de las páginas del web, el html, que se encuentra actualmente en su cuarta revisión parece que pronto será substituido por el xml o html extendido.

En este sentido, alguien atinadamente recordaba lo que alguna vez sucedió en los primeros años de la empresa Intel, cuando un empleado se acercó al director Andy Grove para sugerirle que la compañía comenzara a producir computadoras personales basadas en sus circuitos integrados.

Escépticamente, Grove le preguntó al empleado qué se podría hacer con una computadora personal. El empleado, tratando de encontrar un buen ejemplo le dijo que se podrían utilizar para guardar recetas de cocina.

A Grove le pareció una mala idea, ésta, la de hacer computadoras personales, y no porque fuera un individuo carente de visión.

Simplemente, en ese momento, nadie pensaba en sustituir los recetarios de cocina, ni las agendas y cuadernos, ni el lápiz y el papel, ni la calculadora, ni la máquina de escribir, ni los sobres para carta, ni las reglas y escuadras de dibujo, etc. En fin, nadie pensaba en los alcances de la computadora personal.

Un suceso similar se ha presentado con respecto al Internet. Al principio había muchos escépticos, pero a medida que ha transcurrido el tiempo desde su invención, se ha demostrado su gran utilidad en cualquier ámbito.

Así, que no es de sorprender la enorme velocidad de desarrollo que han alcanzado las tecnologías orientadas al web, y que por lo menos en unas décadas más no parará de crecer y crecer.

El presente trabajo representa el final del ciclo universitario, por lo cual es pertinente reflexionar un poco acerca del impacto de la vida universitaria en mi vida profesional.

La función principal de la universidad como institución, no es crear y transmitir la ciencia y la cultura, ni formar para la vida profesional... su función principal es, desde mi punto de vista, como toda institución educativa, enseñar al ser humano a ser él mismo y convivir con los demás.

Fueron varios los años y muchas las materias que cursé a lo largo de la carrera (Ingeniería en computación), y la misma cantidad de profesores que me instruyeron; es muy cierto, aprendí muchísimas cosas, otras ya las olvidé. Es gracioso como hace algunos días revisando mi certificado de estudios (en el cual aparece un listado de todas las asignaturas cursadas) de repente me pregunté “¿De verdad tomé ésta materia?, ¡porque no la recuerdo!”. Pero que más da, muchos conocimientos así como llegaron se fueron, sin embargo, lo más importante que aprendí en la universidad nunca se me olvidará porque me servirá para ser la mejor en el área que yo desee: Ser autodidacta y compartir mis conocimientos.

Ahora puedo asegurar que si bien no conozco todas las respuestas a los problemas que enfrento en mi trabajo, si sé cómo buscar y encontrar las soluciones. Es muy cierta la frase: “Los triunfadores no son aquellos que lo saben todo, sino quienes lo dan todo por saber y comparten todo lo que saben”.

BIBLIOGRAFÍA

LIBROS Y ARTÍCULOS

Patric Naughton, Herbert Shildt.
Java Manual de Referencia.
McGraw-Hill (1997).

Bruce Eckel. Piensa en Java.
Edicion: 2ª ed.
Prentice Hall , (2002)

Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazález, Alberto Larzabal, Jesús Calleja y Jon García.
Aprenda Java como si estuviera en primero.
Universidad de Navarra. 1999.

Joyanes, L.
Programación orientada a objetos (2ª Ed.),
Prentice-Hall, 1998

Pedro Manuel Cuenca Jiménez.
Programación en Java para Internet.
Anaya Multimedia. 1996.

Grady Booch, James Rumbaugh, Ivar Jacobson
El Lenguaje Unificado de Modelado
Ed. Addison Wesley, Madrid,
1999

Ivar Jacobson, Grady Booch, James Rumbaugh
El Proceso Unificado de Desarrollo de Software
Ed. Addison Wesley, Madrid, 2000

Larman, C.
UML y patrones : introducción al análisis y diseño orientado a objetos
Ed. Prentice Hall, 1999

Ted Husted, Cedric Dumoulin, George Franciscus, y David Winterfeldt.
Struts in Action.
Ed. Manning Publications. 2003.

Chuck Cavaness.
Programming Jakarta Struts.
Ed. O'Reilly. 2002.

Altadill Izura, Pello Xavier.
Struts. Implementación del patrón MVC en aplicaciones Web,
http://cursosgratis.emagister.com/frame.cfm?id_user=151231403052006315656484954695

07&id_centro=579530300529575648666666952674548&id_curso=63658060052150655255657053494568&url_frame=http://www.emagister.com/public/pdf/comunidad_emagister/STRUTS.pdf, 15 de mayo de 2006.

Larman, Craig.
UML y Patrones. Introducción al análisis y diseño orientado a objetos" 2da Edición (traducción de la edición original en ingles "Appling UML and patterns. An introduction to the object oriented analysis and design")
Editorial Prentice Hall Hispanoamericana, S.A. México, 1999.

PÁGINAS WEB

http://java.sun.com
Página oficial de Sun Microsystems (Java).

http://jakarta.apache.org/struts
Página oficial de la Apache Foundation (Proyecto Jakarta).

http://es.wikipedia.org
Enciclopedia virtual para múltiples consultas (JSP, HTML, Ingeniería de Software, etc).

http://www.dte.upct.es/investigacion/is/UML2.0.pdf
Manual de UML.

http://www.db.informatik.uni-bremen.de/umlbib/
Concentrado de bibliografía sobre UML.

http://www.csslab.cl/
Hojas de Estilo (CSS).

http://www.javalobby.org/articles/struts/?source=archives
JSP y Struts.

http://www.desarrolloweb.com/articulos/656.php
Javascript.

http://www.adictosaltrabajo.com/indexg.php?pagina=tutoriales
Tutoriales de desarrollo web (J2EE, Struts, POO, Patrones de Diseño).
http://www.programacion.net/java/tutorial/jdbc/
Acceso a Bases de Datos (JDBC).

http://struts.apache.org/1.x/struts-taglib/tlddoc/html/
Tags HTML de Struts.

http://www.sidar.org/recur/desdi/traduc/es/css/tables.html
Tablas en HTML y Hojas de Estilo.

http://www.desarrolloweb.com/articulos/808.php
Jerarquía de Objetos en Javascript.

http://es.wikipedia.org/wiki/Javadoc
Documentación en java (Javadoc).

http://www.javahispano.org/articles.print.action?id=37
Javadoc.

http://usuarios.lycos.es/dhtml_club/objetos/propiedades.htm
Hojas de Estilo.

http://www.programacion.net/bbdd/articulo/bbdd_disenyo/
Diseño de Bases de Datos.

http://www.programacion.net/java/tutorial/servlets_jsp/
Servlets y JSP.

http://www.programacion.com/bbdd/articulo/ale_poolstruts/
Pool de Conexiones en Struts.

http://www3.uji.es/~mmarques/f47/apun/node83.html
Modelo Entidad Relación.

http://www.geocities.com/trescapas/TresCapas.htm
Modelo de Tres Capas.

http://www.monografias.com/trabajos5/norbad/norbad.shtml
Normalización de Bases de Datos.

http://es.wikipedia.org/wiki/Tomcat
Servidor de aplicaciones Tomcat.