UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

"SISTEMA CON ARQUITECTURA CLIENTE-SERVIDOR DISTRIBUIDA PARA EL LABORATORIO DE COMPUTO-DIE"

T E S I S P R O F E S I O N A L

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

CARLOS ALBERTO ROMAN ZAMITIZ

DIRECTOR DE TESIS: ING. JUAN JOSE CARREON GRANADOS

CIUDAD UNIVERSITARIA, MEXICO, D. F. DICIEMBRE DE 1999

ÍNDICE

1. Introducción

El sistema aquí presentado es un sistema que se ejecuta desde el World Wide Web (WWW) y que permite que los usuarios (alumnos y profesores) del laboratorio de cómputo de la División de Ingeniería Eléctrica (DIE) de la Facultad de Ingeniería puedan conectarse remotamente y poder así consultar qué computadoras están siendo ocupadas y cuáles están libres o desocupadas. Así mismo los usuarios pueden apartar una computadora a un intervalo de tiempo deseado y realizar las operaciones relacionadas con un apartado tales como consultar y cancelar su apartado.

Para la realización del sistema se empezó a utilizar el método OMT para la realización del análisis y el diseño pero posteriormente se investigó más sobre UML que es actualmente un estándar para el análisis y diseño de sistemas. Se analizaron sus ventajas, características y alcances de tal manera que se optó por éste sobre OMT. Además se contó con una excelente bibliografía sobre UML que sirvió para una buena realización del capítulo 3.

Después, en el capítulo 4, se empezó a realizar la justificación del por qué utilizar el lenguaje de programación Java. Se presenta una breve introducción a las bases de datos, qué son, qué tipos hay, qué características tienen, etc., y posteriormente se explica qué es el Sistema Manejador de Bases de Datos de Sybase. Tanto del lenguaje Java como del sistema manejador de bases de datos de Sybase se presentan sus ventajas, alcances y características a fin de que la propuesta de solución y el sistema aquí presentados sean aceptados por el lector.

Posteriormente, en el capítulo 5, se presenta una pequeña introducción de la arquitectura Cliente-Servidor a fin de unificar conceptos con el lector y así sea más comprensible el tema referente a la arquitectura Cliente-Servidor distribuida. De igual manera se explica cómo se comunica el lenguaje Java con los sistemas manejadores de bases de datos. En este capítulo se explica también cómo el sistema se ha integrado con un sistema existente y que anteriormente era el único sistema que se utilizaba para el apartado de computadoras del laboratorio. En un principio el sistema aquí presentado iba a desplazar al sistema existente pero durante el desarrollo, el creador del otro sistema planteó la posibilidad de la integración de ambos sistemas con lo cual se extendió el objetivo de este proyecto y es muy satisfactorio que este sistema haya servido de incentivo para que el sistema ya existente se modificara al grado de que ambos sistemas sirven a la comunidad de la Facultad de Ingeniería.

El capítulo 6 contiene fragmentos de código de los programas que se construyeron para este sistema a fin de exponer el código importante que realiza una cierta función específica. En un principio se pensó en colocar los programas completos pero posteriormente se decidió colocar sólo fragmentos de éstos a fin de crear un sitio web donde se colocarán los programas completos y se reciban sugerencias del funcionamiento del sistema.

El siguiente capítulo explica los requerimientos del sistema así como su instalación y pruebas que se realizaron para posteriormente colocar el sistema en un período de producción.

En el capítulo 8 se presentan las conclusiones que se obtuvieron de este desarrollo. En los capítulos 9 y 10 se presentan las notas que se obtuvieron de libros investigados y la bibliografía, respectivamente.

Creo que el objetivo de este sistema se ha cubierto y la principal intención de este proy es dar la pauta inicial para que, en un futuro no muy lejano, se realice u stema v a Internet para las inscripciones de los alumnos de la Facultad de Ingeniería. De ra, la Facultad confirmará que se encuentra a la altura de las necesidades del país en mo a educación y tecnología se refiere.

2. Definición del Problema

El Laboratorio de Cómputo de la División de Ingeniería Eléctrica (DIE) de la Facultad de Ingeniería de la UNAM cuenta, actualmente, con 2 salas de computadoras. Una de ellas, la sala A con 13 estaciones de trabajo con sistema operativo HP-UX y la otra, la sala B, cuenta con 44 computadoras personales (PC's) que se encuentran en una red con topología de bus con un servidor Novell Intranetware versión 4.1. Estos equipos cuentan con procesador Intel 80486, una velocidad promedio de 66 MHz., y una memoria RAM con capacidad de 16 MB.

La sala B de este laboratorio da servicio, aproximadamente, a 300 alumnos de la Facultad de Ingeniería al día y por lo tanto se requirió, inicialmente, de unas formas en hojas de papel en las cuales se tenían en columnas todas las horas de servicio del laboratorio y en renglones los números de las computadoras. De esta manera, en cada una de las celdas de la tabla, se colocaba el login del usuario abarcando las horas en que este alumno iba a ocupar esa computadora (Figura 2-1).

Posteriormente, se analizó, diseñó y desarrolló un sistema que realizaba los apartados de las computadoras personales que se encuentran en la sala B. Este sistema se desarrolló en lenguaje C utilizando el compilador de Borland versión 2.0 y es el sistema que hasta la fecha se sigue utilizando para registrar el uso y apartado de esas computadoras (Figura 2-2). Sin embargo el sistema presenta las siguientes deficiencias:

- No hace uso de una interfaz gráfica que sea amigable al usuario.
- El almacenamiento de la información se realiza en archivos "planos".
- El sistema no trabaja en ambiente multiprocesos (multithreading).

Debido a que en cada semestre escolar el número de usuarios de este laboratorio va en aumento, se tiene la necesidad de crear un sistema que resuelva las deficiencias anteriormente mencionadas.

FACULTAD DE INGENIERÍA
DIVISIÓN DE INGENIERÍA ELÉCTRICA
DEPARTAMENTO DE COMPUTACIÓN

Fecha:_____ Sistema de apartado para sesiones en la sala **B**.

	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20	20-21	21-22
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															

Figura 2-1

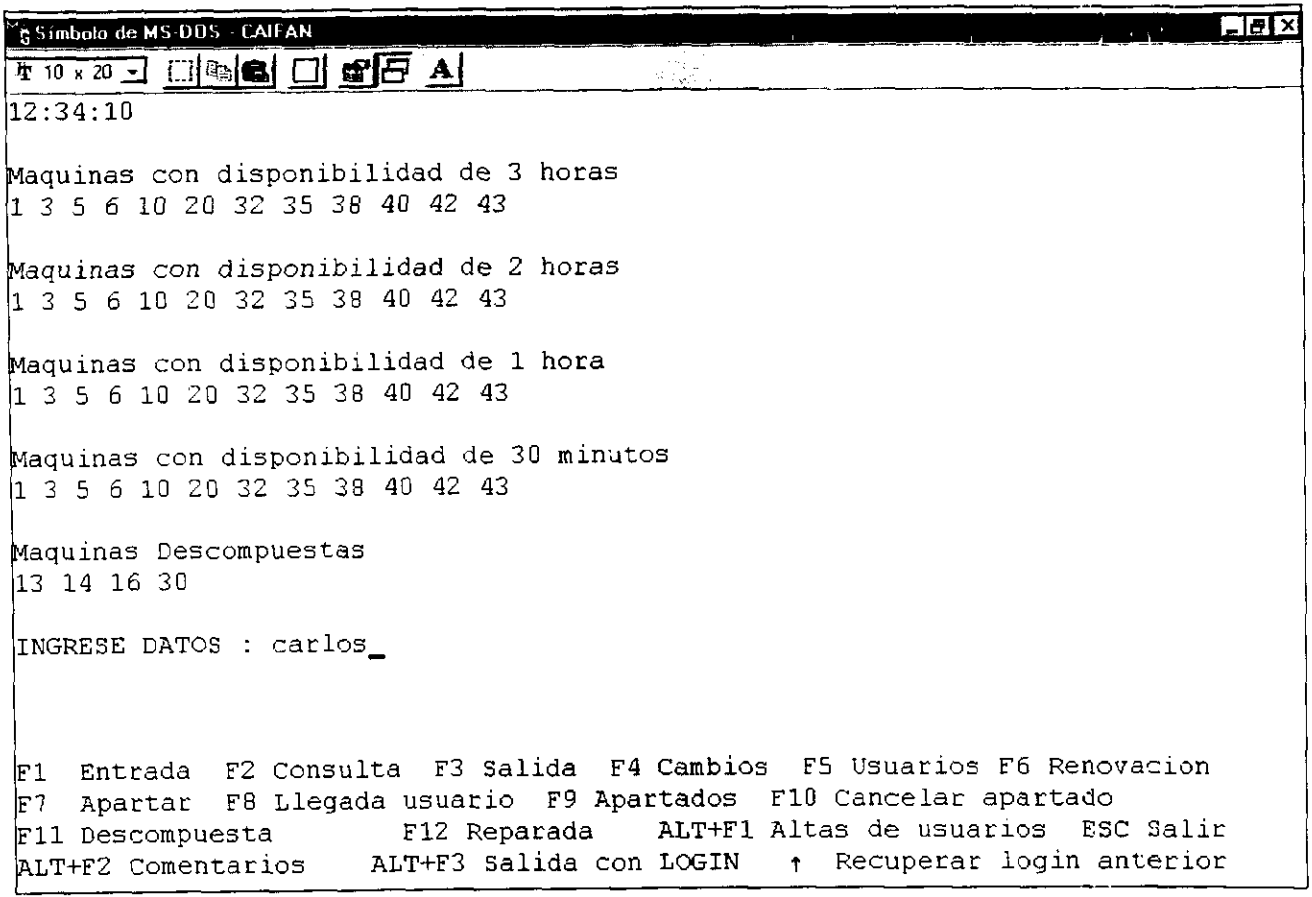

Símbolo de MS-DOS - CAIFAN
10 x 20
12:34:10
Maquinas con disponibilidad de 3 horas
1 3 5 6 10 20 32 35 38 40 42 43
Maquinas con disponibilidad de 2 horas
1 3 5 6 10 20 32 35 38 40 42 43
Maquinas con disponibilidad de 1 hora
1 3 5 6 10 20 32 35 38 40 42 43
Maquinas con disponibilidad de 30 minutos
1 3 5 6 10 20 32 35 38 40 42 43
Maquinas Descompuestas
13 14 16 30
INGRESE DATOS : carlos
F1 Entrada F2 Consulta F3 Salida F4 Cambios F5 Usuarios F6 Renovacion
F7 Apartar F8 Llegada usuario F9 Apartados F10 Cancelar apartado
F11 Descompuesta F12 Reparada ALT+F1 Altas de usuarios ESC Salir
ALT+F2 Comentarios ALT+F3 Salida con LOGIN ↑ Recuperar login anterior

Figura 2-2

3. El Análisis y el Diseño

Para la realización de un sistema se requiere de paradigmas que ayuden a los creadores de ese sistema a analizarlo y diseñarlo pero, ¿Qué es un paradigma? Se tienen varias definiciones "informales":

- Un modelo mental.
- Un punto de vista.
- Un marco de referencia.
- Un marco de trabajo de pensamientos o creencias a través del cual se interpretan el mundo o la realidad.
- Un ejemplo utilizado para definir un fenómeno.
- Una creencia sostenida en común por un grupo de personas, tales como científicos de una disciplina dada o expertos en la materia.

Actualmente, el uso de la palabra *paradigma* tiene sus raíces en la definición dada por el científico y filósofo Thomas Kuhn en su libro *The Structure of Scientific Revolutions* en el año de 1962:

Un paradigma es "*... una constelación de conceptos, valores, percepciones y prácticas compartidas por una comunidad que se forma una visión particular de la realidad y que es la base de la manera en que la comunidad se organiza a sí misma*" [1].

En el ámbito de la Ingeniería de Software, los principales paradigmas que predominan, y en los cuales se pensó para construir este sistema, son:

- El Paradigma Estructurado Procedural.
- El Paradigma Orientado a Objetos.

La primera opción, a pesar de que ha sido empleada durante mucho tiempo en proyectos realmente complejos, presenta tres grandes deficiencias según la mayoría de los ingenieros de software:

- Los productos que resultan al emplear estas técnicas son pocos flexibles.
- Los programadores que lo utilizan tienden a concentrarse en el diseño y la puesta en práctica del sistema, sin tomar en cuenta su vida posterior (el mantenimiento).

- No alientan al programador a aprovechar el trabajo de proyectos anteriores.

La desventaja de utilizar un paradigma que se concentra en el diseño inicial del sistema se hace evidente si se toma en cuenta que la vida útil de un producto de software puede ser cinco o seis veces más grande que el lapso en que se desarrolla. La fase de mantenimiento de un sistema es tan importante que cualquier método de diseño debe tener como objetivo principal producir sistemas que faciliten su propio mantenimiento.

La reutilización de código es otro de los factores que no se toma en cuenta para los métodos de diseño tradicionales. Cualquier programador que desarrolla un sistema nuevo debe escribir una buena cantidad de código que ha escrito anteriormente una y otra vez. Las rutinas de búsqueda, ordenamiento, manejo de menús y despliegue de ventanas, entre otras, se repiten continuamente en proyectos diferentes. Los métodos de desarrollo de software deben alentar al programador a utilizar el código escrito previamente ya sea por él mismo o por otros programadores.

Tradicionalmente, se han empleado tres tipos de reutilización: de personal, de diseño y de código fuente. En el primer tipo, a un proyecto nuevo se le asignan programadores que tienen experiencia en el desarrollo de proyectos semejantes; el segundo tipo consiste en emplear el diseño de un sistema para desarrollar otro sistema similar y el tercer tipo se da cuando un programador utiliza parte de un programa escrito con anterioridad para crear un nuevo programa. Sin embargo, estas tres formas de reutilización se han empleado en forma muy limitada, por lo que hay que buscar paradigmas más generales, entre los que destaca el *Paradigma Orientado a Objetos*.

3.1 Comparación entre el Paradigma Estructurado y el Paradigma Orientado a Objetos

Actualmente, ambos paradigmas son muy utilizados en la construcción de sistemas de software, pero el Paradigma Orientado a Objetos se está imponiendo sobre los demás para la creación de sistemas de software y se ha convertido en el estándar, pero ¿Qué nos ofrece cada uno de estos dos paradigmas?. En el cuadro siguiente (Figura 3-1) se muestran algunas de las características con las que cuenta cada paradigma.

Paradigma *Estructurado* Procedural	**Paradigma Orientado a Objetos**
El mundo real es representado por entidades lógicas y control de flujo.	El mundo real es representado por objetos que imitan a las entidades externas.
Utiliza abstracción procedural.	Utiliza abstracción de clases y objetos.
Unidad de estructura: sentencia o expresión.	Unidad de estructura: objeto tratado como componente de software.
Los sistemas se construyen basándose en el fundamento de las funciones.	Los sistemas se construyen basándose en el principio de las estructuras de datos (objetos).
Utiliza descomposición funcional.	Utiliza descomposición orientada a objetos.
Estructuras de datos pasivas y tontas.	Estructuras de datos activas e inteligentes (objetos) encapsulan todos los procedimientos pasivos.
En un programa, los datos y los procedimientos están separados.	En un programa, el estado de los objetos (tipos de datos) y el comportamiento (operaciones) están encapsulados.
Los programas son ligados a través del mecanismo de paso de parámetros y el sistema operativo.	Los programas son partes integradas del programa completo. Un programa es una colección de objetos que interactúan.
Los programadores son responsables de llamar los procedimientos activos para el paso de parámetros.	Los objetos activos se comunican con otros pasando mensajes para activar sus operaciones.
El programador debe estar seguro de que el procedimiento trabajará correctamente sobre el tipo de dato al cual se aplica.	El paso de mensajes asegura que se puede tener acceso al estado interno de un objeto solamente si es permitido, así como la encapsulación previene el acceso no autorizado.
Utiliza lenguajes orientados a procedimientos tales como C o Pascal.	Utiliza lenguajes orientados a objetos tales como C++, SmallTalk, Java, etc.

Figura 3-1

3.2 El Paradigma Orientado a Objetos

A medida que se acercaban los años 80, el Paradigma Orientado a Objetos para la Ingeniería de Software comenzaba a madurar como un enfoque de desarrollo de software. Empezamos a crear diseños de aplicaciones de todo tipo utilizando la forma de pensar orientada a los objetos e implementamos (codificamos) programas utilizando lenguajes y técnicas orientadas a los objetos. Sin embargo, el análisis de requisitos, es decir, la relación entre la asignación de software al nivel del sistema y el diseño del software, se quedó atrás.

El Paradigma Orientado a Objetos presenta características que lo hacen idóneo para el análisis, diseño y programación de sistemas. Algunas de las razones para utilizar dicho paradigma se explican en los siguientes párrafos.

El mundo real es un lugar complejo en el sentido de que todas las cosas, tangibles e intangibles, son complejas si las analizamos a partir de sus componentes más básicos. Se dice que el Paradigma Orientado a Objetos es más natural que el tradicional, el Paradigma Estructurado, y es cierto en dos sentidos. En un sentido, dicho paradigma es más natural porque nos permite organizar la información de una forma similar para nosotros. En el otro sentido es más natural porque refleja las técnicas de la naturaleza para manejar la complejidad. Para entender mejor lo anterior, veamos la manera, de forma más general, en que la naturaleza le da poder al concepto de objeto.

El bloque básico de construcción en la naturaleza, y de la cual está compuesta toda la variedad de seres vivos que existimos sobre la Tierra, es la célula. Si observamos la estructura interna de la misma (Figura 3-2), podemos hacer las siguientes conclusiones:

1. La célula está protegida por una membrana que controla el acceso a su interior. De hecho, la membrana encapsula a la célula, protegiendo su forma interna de trabajar de la intromisión externa.
2. La célula contiene formas de manejar información y comportamiento. La mayoría de la información que define su comportamiento se encuentra en el núcleo, en el centro de la célula (DNA y RNA). El comportamiento, por ejemplo la síntesis de proteínas y la conversión de energía, se lleva a cabo en el cuerpo medio de la célula.

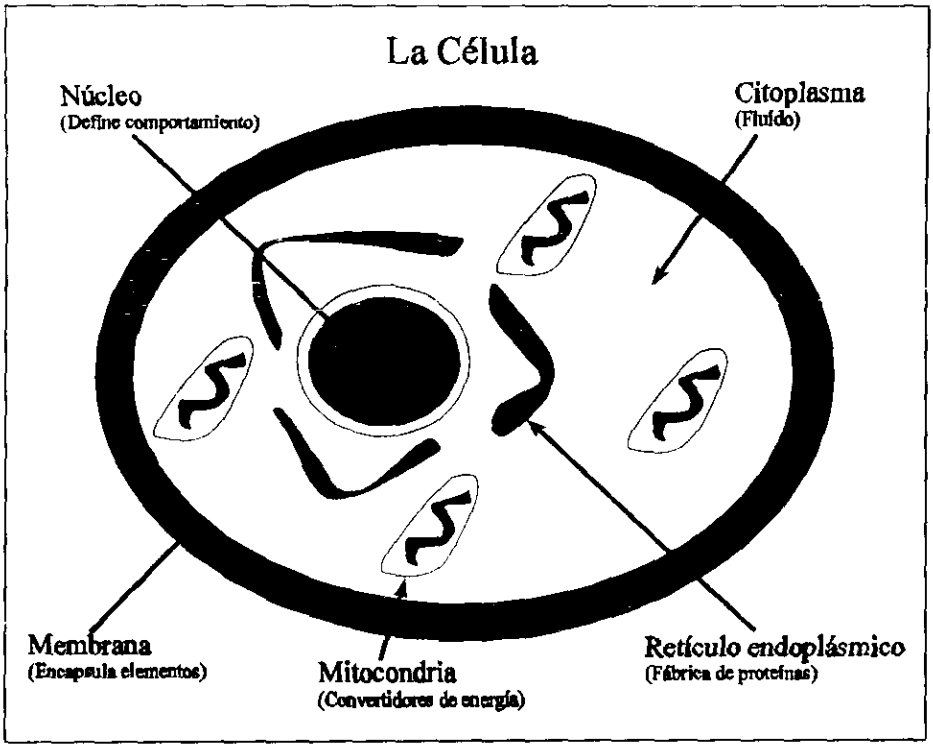


Figura 3-2

La interacción de las células se realiza a través de la membrana ya que es ésta la que permite intercambiar mensajes químicos con otras células. Una célula puede o no permitir el intercambio del mensaje químico. La membrana representa, entonces, su interfaz con el exterior. Esta comunicación basada en mensajes simplifica enormemente la forma en que las células realizan su función. No es necesario que una célula conozca el funcionamiento interno de otra; lo único que tiene que hacer es enviarle el mensaje apropiado para que la célula receptora ejecute esa acción. Cuando una célula muere, por ejemplo, se liberan sustancias (mensajes) con las que otras células se dan cuenta; entonces una de ellas se divide en dos nuevas células para sustituir a la que murió, regenerándose así los tejidos.

Como podemos apreciar, la interacción entre células está basada en mensajes. Esto quiere decir que cuando una célula quiere afectar el comportamiento de otras células, aquélla les envía un mensaje químico que provoca que éstas modifiquen su comportamiento al realizar cierta acción. Cuando una célula recibe un mensaje que tiene significado para ella, responde al estímulo de acuerdo con la información que tiene codificada en su núcleo.

En pocas palabras, las células no interactúan entre sí interfiriendo con el DNA de las otras. Las células se ocupan de sus propias actividades y hacen requerimientos de una manera ordenada, en lugar de tratar de controlar la operación interna de las otras células directamente. Esta situación ofrece dos formas diferentes de protección:

1. Protege el interior de cada célula de ser modificado por otras. Cualquier célula viviente es mucho más compleja que la mayoría de los sistemas de cómputo jamás creados. Al proteger el interior de la célula del mundo externo, mantiene su integridad.
2. Protege a otras células al no conocer el manejo interno de dichas células.

La segunda ventaja es un ejemplo vivo de lo que es el ocultamiento de la información. De hecho, las células funcionan tan bien como módulos en sistemas complejos como los seres vivos, gracias a que protegen al resto del sistema del tener que lidiar con la complejidad interna. Cuando no se respeta esta independencia entre células y la puesta en práctica interna, la célula enferma y muere. Veamos el problema con un virus; el virus se interna en la célula y la ataca desde adentro. Dado que el virus está arraigado en la célula, no puede ser destruido por los glóbulos blanco porque destruiría también a la célula.

Así como la célula del ejemplo anterior, de igual manera es de complejo el mundo que rodea al hombre. Típicamente, el ser humano entiende el mundo que lo rodea por la construcción de modelos mentales, de porciones o fragmentos de ese mundo, para tratar de entender las cosas con las que él interactúa. En esencia, este proceso de construcción de modelos es semejante al de diseño de software.

Además, el modelo mental debe ser más simple que el sistema al que está modelando o dicho modelo será inútil. El ser humano debe abstraerse para poder entender el sistema y realizar una migración, de éste, al modelo mental considerando sólo información cuidadosamente seleccionada e ignorando características irrelevantes. Este proceso de *abstracción* es psicológicamente natural y necesario para que nosotros entendamos el mundo que nos rodea.

3.2.1 La Abstracción como herramienta en el Paradigma Orientado a Objetos

La abstracción es esencial para el funcionamiento de la mente humana, una poderosa herramienta para tratar la complejidad y la llave para diseñar buen software.

Históricamente, los programadores enviaban instrucciones binarias a una computadora agrupando interrupciones en su panel frontal. Los mnemónicos del lenguaje ensamblador eran abstracciones diseñadas para eliminar la necesidad de recordar las secuencias de bits de las cuales estaban compuestas las instrucciones. En cambio, los programadores realizaron una abstracción de esas secuencias de bits y les agregaron un nombre, llegando a un nivel de abstracción muy alto.

El siguiente uso de la abstracción se dio en la habilidad para crear instrucciones definidas por el usuario. El conjunto de instrucciones de una máquina dada podía ser extendido agrupando secuencias de esas instrucciones primitivas en macro-instrucciones y agregándoles un nombre. Un conjunto de instrucciones optimizado podía, entonces, ser invocado por una macro-instrucción. Una sola de las macro-instrucciones puede lograr que una máquina haga muchas cosas.

Los lenguajes de programación de alto nivel permitieron a los programadores separarse de la arquitectura especifica de una computadora dada. Ciertas secuencias de instrucciones fueron reconocidas como universalmente útiles y los programas fueron escritos en términos de éstas. Cada instrucción podía invocar una variedad de instrucciones-máquina, dependiendo de la máquina específica para la cual los programas fueron compilados. Esta abstracción permitió a los programadores escribir software para un propósito genérico, sin preocuparse por la máquina que ejecutaría el programa.

La programación estructurada abarcó el uso de abstracciones de control tales como los ciclos o sentencias if-else que fueron incorporados a lenguajes de alto nivel. Lo anterior permitió a los programadores abstraer condiciones comunes para cambiar la secuencia de la ejecución.

Más recientemente, los tipos de datos abstractos han permitido a los programadores escribir código sin preocuparse por la forma específica en que los datos serán representados. En cambio, los programadores pueden codificar al nivel abstracto en el que pudieran estar realizados los datos. Los detalles de su representación están escondidos; como la estructura de datos (un objeto) es implementada puede considerarse a un detalle de bajo nivel en el que no es necesario preocuparse por quién diseña y estructura el programa. Por ejemplo, un

conjunto puede ser definido como una colección no ordenada de elementos entre los cuales no hay duplicados. Utilizando esta definición, nosotros podemos especificar las operaciones de esa estructura (el objeto) que pueden ser ejecutadas en un conjunto sin especificar si sus elementos están almacenados en un arreglo, una lista ligada o alguna otra estructura de datos (otro objeto).

El *Diseño Orientado a Objetos* (*DOO*) descompone un sistema en objetos, el componente básico del diseño. Este uso de objetos permite la adición de otros mecanismos de abstracción.

3.2.2 Algunos conceptos del Paradigma Orientado a Objetos

De todo lo anterior se puede concluir que el Paradigma Orientado a Objetos ha derivado de los paradigmas anteriores a éste. Así como los métodos de diseño estructurado realizados guían a los desarrolladores que tratan de construir sistemas complejos utilizando algoritmos como sus bloques fundamentales de construcción, similarmente los métodos de diseño orientado a objetos han evolucionado para ayudar a los desarrolladores a explotar el poder de los lenguajes de programación basados en objetos y orientados a objetos, utilizando las clases y objetos como bloques de construcción básicos.

Actualmente el modelo de objetos ha sido influenciado por un número de factores no sólo de la *Programación Orientada a Objetos*, *POO* (*Object Oriented Programming*, *OOP* por sus siglas en inglés). Además, el modelo de objetos ha probado ser un concepto uniforme en las ciencias de la computación, aplicable no sólo a los lenguajes de programación sino también al diseño de interfaces de usuario, bases de datos y arquitectura de computadoras por completo. La razón de ello es, simplemente, que una orientación a objetos nos ayuda a hacer frente a la inherente complejidad de muchos tipos de sistemas.

El Paradigma Orientado a Objetos representa, entonces, un desarrollo evolucionario y no revolucionario. No rompe con los avances del pasado pero se construye sobre éstos. Desafortunadamente, muchos programadores hoy en día están utilizando, formalmente e informalmente, los principios de diseño estructurado por lo que, hasta la fecha, ambos tipos de diseño de sistemas han coexistido.

Debido a que el modelo de objetos deriva de diversas fuentes, ha sido acompañado de una confusión de terminologías, por lo aquí se tratará de unificar conceptos.

Se define a un objeto como "*una entidad tangible que muestra alguna conducta bien definida*" [2]. Un objeto "*es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y los métodos que controlan dichos datos*" [3].

Los objetos tienen una cierta "integridad" la cual no deberá ser violada. En particular, un objeto puede solamente cambiar estado, conducta, ser manipulado o estar en relación con otros objetos de manera apropiada a este objeto.

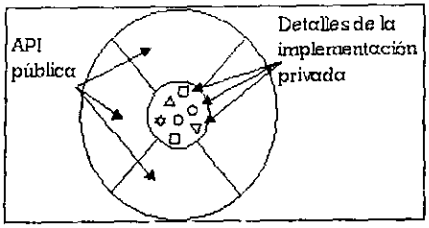
Actualmente, el *Análisis Orientado a Objetos* (*AOO*) va progresando como método de análisis de requisitos por derecho propio y como complemento de otros métodos de análisis. En lugar de examinar un problema mediante el modelo clásico de entrada-proceso-salida (flujo de información) o mediante un modelo derivado exclusivamente de estructuras jerárquicas de información, el AOO introduce varios conceptos nuevos. Estos conceptos nuevos le parecen inusuales a mucha gente, pero son bastante naturales. Considerando este hecho, Coad y Yourdon escriben: "*El AOO —Análisis Orientado a Objetos— se basa en conceptos que una vez aprendimos en la guarderia: objetos y atributos, clases y miembros, todo y parte. Nadie sabe por qué hemos tardado tanto tiempo en aplicar estos conceptos en el análisis y la especificación de los sistemas de información —quizás hemos estado demasiado ocupados 'siguiendo el flujo' durante nuestros análisis estructurados diarios, para ponernos a considerar otras alternativas*" [4].

Algunos conceptos que se utilizan en el Paradigma Orientado a Objetos son los siguientes:

Una *clase* es una plantilla para objetos múltiples con características similares. Las clases comprenden todas esas características de un conjunto particular de objetos. Cuando se escribe un programa en lenguaje orientado a objetos, no se definen objetos verdaderos sino se definen clases de objetos.

Una *instancia* de un objeto es otro término para un objeto real. Si la clase es la representación general de un objeto, una instancia es su representación concreta. A menudo se utiliza indistintamente la palabra objeto o instancia para referirse, precisamente, a un objeto.

En los lenguajes orientados a objetos, cada clase está compuesta de dos cualidades: *atributos* (estado) y *métodos* (comportamiento o conducta) como se observa en la figura 3-3 en la que dichos atributos están en la parte interna de la clase y los métodos se encuentran en la parte externa "cubriendo" los atributos.

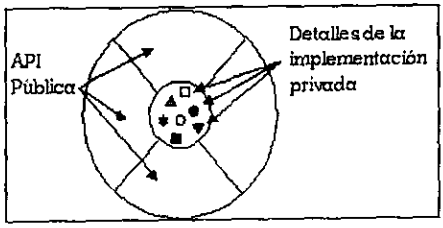


Representación visual de una clase como componente de software

Figura 3-3

Los atributos son las características individuales que diferencian a un objeto de otro y determinan la apariencia, estado u otras cualidades de ese objeto. Los atributos de un objeto incluyen información sobre su estado.

Los métodos de una clase determinan el comportamiento o conducta que requiere esa clase para que sus instancias puedan cambiar su estado interno o cuando dichas instancias son llamadas para realizar algo por otra clase o instancia. El comportamiento es la única manera en que las instancias pueden hacerse algo a sí mismas o tener que hacerles algo (Figura 3-4). Al igual que en la figura 3-3, los atributos se encuentran en la parte interna mientras que los métodos se encuentran en la parte externa del objeto.



Representación visual de un objeto como componente de software

Figura 3-4

Para definir el comportamiento de un objeto, se crean métodos, los cuales tienen una apariencia y un comportamiento igual al de las funciones en otros lenguajes de programación, los lenguajes estructurados, pero se definen dentro de una clase. Los métodos no siempre afectan a un solo objeto; los objetos también se comunican entre sí mediante el uso de métodos. Una clase u objeto puede llamar métodos en otra clase u objeto para avisar sobre los cambios en el ambiente o para solicitarle a ese objeto que cambie su estado.

Cualquier cosa que un objeto no sabe, o no puede hacer, es excluida del objeto. Además, como se puede observar de los diagramas, las variables del objeto se localizan en el centro o núcleo del objeto. Los métodos rodean y esconden el núcleo del objeto de otros objetos en el programa. Al empaquetamiento de las variables de un objeto con la protección de sus métodos se le llama *encapsulamiento*. Típicamente, el encapsulamiento es utilizado para esconder detalles de la puesta en práctica no importantes de otros objetos. Entonces, los detalles de la puesta en práctica pueden cambiar en cualquier tiempo sin afectar otras partes del programa.

Esta imagen conceptual de un objeto —un núcleo de variables empaquetadas en una membrana protectora de métodos— es una representación ideal de un objeto y es el ideal por el que los diseñadores de sistemas orientados a objetos luchan. Sin embargo, no lo es todo: a menudo, por razones de eficiencia o la puesta en práctica, un objeto puede querer exponer algunas de sus variables o esconder algunos de sus métodos.

El encapsulamiento de variables y métodos en un componente de software ordenado es, todavía, una simple idea poderosa que provee dos principales beneficios a los desarrolladores de software:

- *Modularidad*, esto es, el código fuente de un objeto puede ser escrito, así como darle mantenimiento, independientemente del código fuente de otros objetos. Así mismo, un objeto puede ser transferido alrededor del sistema sin alterar su estado y conducta.
- *Ocultamiento de la información*, es decir, un objeto tiene una "interfaz publica" que otros objetos pueden utilizar para comunicarse con él. Pero el objeto puede mantener información y métodos privados que pueden ser cambiados en cualquier tiempo sin afectar a los otros objetos que dependan de ello.

En las ilustraciones anteriores de una clase y un objeto, figuras 3-3 y 3-4, se observa que son muy similares una de la otra y puede prestarse a una seria confusión. En el mundo real, es obvio que las clases mismas no son los objetos que ellas describen o contienen. Sin embargo, es un poco más difícil diferenciar clases y objetos en software porque los objetos de software son, simplemente, modelos electrónicos de objetos del mundo real o conceptos abstractos en primer lugar. Lo anterior sucede porque mucha gente utiliza el término "objeto" irregularmente y lo hace para referirse tanto a clases como instancias.

La figura anterior de la clase, figura 3-3, no está "sombreada" porque representa un "proyecto", general y detallado, de un objeto en lugar de un objeto mismo. En comparación, un objeto está sombreado indicando que ese objeto actualmente existe y puede ser utilizado.

Los objetos proveen el beneficio de la modularidad y el ocultamiento de la información. Las clases proveen el beneficio de la reutilización. Los programadores de software utilizan la misma clase, y por lo tanto el mismo código, una y otra vez para crear muchos objetos.

En las implantaciones orientadas a objetos se percibe un objeto como un paquete de datos y procedimientos que se pueden llevar a cabo con estos datos. Esto encapsula los datos y los procedimientos. La realidad es diferente: los atributos se relacionan al objeto o instancia y los métodos a la clase. ¿Por qué se hace así? Los atributos son variables comunes en cada objeto de una clase y cada uno de ellos puede tener un valor asociado, para cada variable, diferente al que tienen para esa misma variable los demás objetos. Los métodos, por su parte, pertenecen a la clase y no se almacenan en cada objeto, puesto que sería un desperdicio almacenar el mismo procedimiento varias veces y ello va contra el principio de reutilización de código.

Otro concepto muy importante en el Paradigma Orientado a Objetos es el de *herencia*. La herencia es un mecanismo poderoso con el cual se puede definir una clase en términos de otra clase; lo que significa que cuando se escribe una clase, sólo se tiene que especificar la diferencia de esa clase con otra, con lo cual, la herencia dará acceso automático a la información contenida en esa otra clase.

Con la herencia, todas las clases están arregladas dentro de una jerarquía estricta. Cada clase tiene una superclase (la clase superior en la jerarquía) y puede tener una o más subclases (las clases que se encuentran debajo de esa clase en la jerarquía). Se dice que las clases inferiores en la jerarquía, las clases hijas, heredan de las clases más altas, las clases padres.

Las subclases heredan todos los métodos y variables de las superclases. Es decir, en alguna clase, si la superclase define un comportamiento que la clase hija necesita, no se tendrá que redefinir o copiar ese código de la clase padre (Figura 3-5).

De esta manera, se puede pensar en una jerarquía de clase como la definición de conceptos demasiado abstractos en lo alto de la jerarquía y esas ideas se convierten en algo más concreto conforme se desciende por la cadena de la superclase.

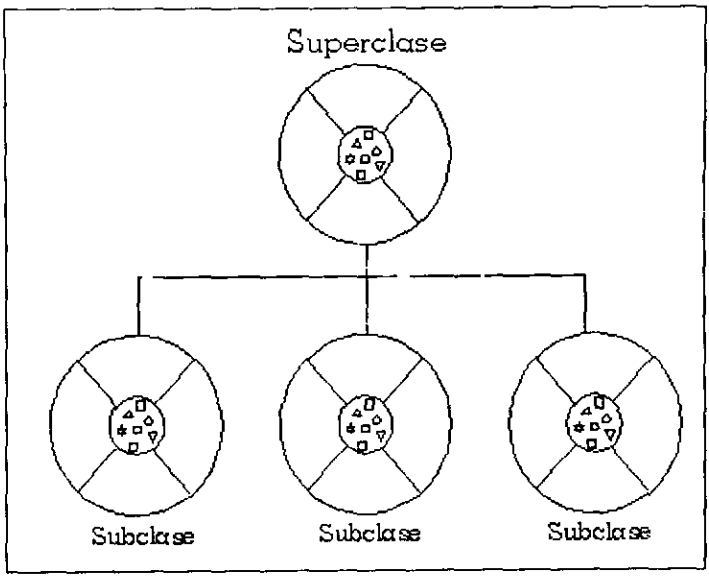


Figura 3-5

Sin embargo, las clases hijas no están limitadas al estado y conducta provistos por sus superclases; pueden agregar variables y métodos además de los que ya heredan de sus clases padres. Las clases hijas pueden, también, sobreescribir los métodos que heredan por implementaciones especializadas para esos métodos. De igual manera, no hay limitación a un sólo nivel de herencia por lo que se tiene un árbol de herencia en el que se puede heredar varios niveles hacia abajo y mientras más niveles descienda una clase, más especializada será su conducta.

La herencia presenta los siguientes beneficios:

- Las subclases proveen conductas especializadas sobre la base de elementos comunes provistos por la superclase. A través del uso de herencia, los programadores pueden reutilizar el código de la superclase muchas veces.
- Los programadores pueden implementar superclases llamadas clases abstractas que definen conductas "genéricas". Las superclases abstractas definen, y pueden implementar parcialmente, la conducta pero gran parte de la clase no está definida ni implementada. Otros programadores concluirán esos detalles con subclases especializadas.

3.2.3 Ventajas del Paradigma Orientado a Objetos

En síntesis, algunas ventajas que presenta el Paradigma Orientado a Objetos son:

- *Reutilización.* Las clases están diseñadas para que se reutilicen en muchos sistemas. Para maximizar la reutilización, las clases se construyen de manera que se puedan adaptar a los otros sistemas. Un objetivo fundamental de las técnicas orientadas a objetos es lograr la reutilización masiva al construir el software.

- *Estabilidad.* Las clases diseñadas para una reutilización repetida se vuelven estables, de la misma manera que los microprocesadores y otros chips se hacen estables.

- *El diseñador piensa en términos del comportamiento de objetos y no en detalles de bajo nivel.* El encapsulamiento oculta los detalles y hace que las clases complejas sean fáciles de utilizar.

- *Se construyen clases cada vez más complejas.* Se construyen clases a partir de otras clases, las cuales a su vez se integran mediante clases. Esto permite construir componentes de software complejos, que a su vez se convierten en bloques de construcción de software más complejo.

- *Calidad.* Los diseños suelen tener mayor calidad, puesto que se integran a partir de componentes probados, que han sido verificados y pulidos varias veces.

- *Un diseño más rápido.* Las aplicaciones se crean a partir de componentes ya existentes. Muchos de los componentes están construidos de modo que se pueden adaptar para un diseño particular.

- *Integridad.* Las estructuras de datos (los objetos) sólo se pueden utilizar con métodos específicos. Esto tiene particular importancia en los sistemas cliente-servidor y los sistemas distribuidos, en los que usuarios desconocidos podrían intentar el acceso al sistema.

- *Mantenimiento más sencillo.* El programador encargado del mantenimiento cambia un método de clase a la vez. Cada clase efectúa sus funciones independientemente de las demás.

- *Una interfaz de pantalla sugestiva para el usuario.* Hay que utilizar una interfaz de usuario gráfica de modo que el usuario apunte a iconos o elementos de un menú desplegado, relacionados con los objetos. En determinadas ocasiones, el usuario puede ver un objeto en la pantalla. Ver y apuntar es más fácil que recordar y escribir.

- *Independencia del diseño.* Las clases están diseñadas para ser independientes del ambiente de plataformas, hardware y software. Utilizan solicitudes y respuestas con formato estándar. Esto les permite ser utilizadas en múltiples sistemas operativos, controladores de bases de datos, controladores de red, interfaces de usuario gráficas, etc. El creador del software no tiene que preocuparse por el ambiente o esperar a que éste se especifique.

- *Interacción.* El software de varios proveedores puede funcionar como conjunto. Un proveedor utiliza clases de otros. Existe una forma estándar de localizar clases e interactuar con ellas. El software desarrollado de manera independiente en lugares ajenos debe poder funcionar en forma conjunta y aparecer como una sola unidad ante el usuario.

- *Computación Cliente-Servidor.* En los sistemas cliente-servidor, las clases en el software cliente deben enviar solicitudes a las clases en el software servidor y recibir respuestas. Una clase servidor puede ser utilizada por clientes diferentes. Estos clientes sólo pueden tener acceso a los datos del servidor a través de los métodos de la clase. Por lo tanto los datos están protegidos contra su corrupción.

- *Computación de distribución masiva.* Las redes a nivel mundial utilizarán directorios de software de objetos accesibles. El diseño orientado a objetos es la clave para la computación de distribución masiva. Las clases de una máquina interactúan con las de algún otro lugar sin saber donde residen tales clases. Ellas reciben y envían mensajes orientados a objetos en formato estándar.

- *Mayor nivel de automatización de las bases de datos.* Las estructuras de datos (los objetos) en las bases de datos orientadas a objetos están ligadas a métodos que llevan a cabo acciones automáticas. Una base de datos OO tiene integrada una *inteligencia*, en forma de métodos, en tanto que una base de datos relacional básica carece de ello.

- *Migración.* Las aplicaciones ya existentes, sean orientadas a objetos o no, pueden preservarse si se ajustan a un contenedor orientado a objetos, de modo que la comunicación con ella sea a través de mensajes estándar orientados a objetos.

- *Mejores herramientas CASE.* Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) utilizarán las técnicas gráficas para el diseño de las clases y de la interacción entre ellas, para el uso de los objetos existentes adaptados a nuevas aplicaciones. Las herramientas deben facilitar el modelado en términos de eventos, formas de activación, estados de objetos, etc. Las herramientas OO del CASE deben generar un código tan pronto se definan las clases y permitir al diseñador utilizar y probar los métodos recién creados. Las herramientas se deben diseñar de manera que apoyen el máximo de creatividad y una continua afinación del diseño durante la construcción.

Dentro del Paradigma Orientado a Objetos existen muchos métodos de modelado de sistemas, de los cuales el predominante es *OMT* (*Object Modeling Technique, Tecnica de Modelado de Objetos*) pero en 1995 surgió no un método sino un lenguaje de modelado de sistemas llamado *UML* (*Unified Modeling Language. Lenguaje Unificado de Modelado*).

3.2.4 Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML) es una técnica para la especificación de sistemas en todas sus fases. Ha sido desarrollado por los más importantes autores en materia de Análisis y Diseño de Sistemas y ha sido utilizado con éxito en sistemas construidos para toda clase de industrias alrededor del mundo: hospitales, bancos, comunicaciones, aeronáutica, finanzas, etc.

UML fue creado por los autores de las metodologías más prominentes de los últimos 10 años: James Rumbaugh, autor de OMT, Ivar Jacobson, autor de OOSE (Object Oriented Software Engineering) y Grady Booch, autor del Método Booch.. El proyecto UML fue auspiciado por Rational Software Corporation y arrancó oficialmente en Octubre de 1994. La versión 0.8 se liberó en Octubre de 1995 y para Enero de 1997 se liberó oficialmente la versión 1.0 con la colaboración de Digital Equipment Corporation, Hewlett-Packard, I-Logix., Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational, Texas Instruments y Unisys. De esta colaboración surgió un lenguaje de modelado bien definido, expresivo, poderoso y aplicable a un amplio espectros de dominios de problemas. UML 1.0 fue ofrecido al *Object Management Group* (*OMG*) en Enero de 1997

como respuesta a su requerimiento de una propuesta para un lenguaje estándar de modelado. UML 1.1 fue adoptado como estándar por el OMG el 14 de Noviembre de 1997.

UML sirve para hacer modelos que permitan:

- Visualizar cómo es un sistema o cómo queremos que sea.
- Especificar la estructura y/o comportamiento de un sistema.
- Hacer una plantilla que guíe la construcción de los sistemas.
- Documentar las decisiones que se han tomado.

El modelado sirve no solamente para los grandes sistemas, aun en aplicaciones de pequeño tamaño se obtienen beneficios de modelado, sin embargo es un hecho que entre más grande y más complejo es el sistema, más importante es el papel que juega el modelado por una simple razón: "El hombre hace modelos de sistemas complejos porque no puede entenderlos en su totalidad".

Hay límites para el entendimiento de la complejidad. A través del modelado reducimos el ámbito del problema de estudio al enfocar sólo un aspecto a la vez.

UML consiste de:

- Reglas de simbología que aplican a cualquier tipo de modelo hecho bajo este lenguaje, por ejemplo, el modo en que se coloca un comentario en cualquier diagrama o el modo en que se aumenta la nomenclatura existente en UML.
- Diferentes tipos de diagramas: Diagramas Uses-Case, Diagramas de clase, Diagramas de Componentes, Diagramas de Distribución, etc. Cada diagrama está diseñado para enfocar un aspecto en particular de un sistema.

UML soporta prácticamente todas las fases de un ciclo de desarrollo de un sistema: Recopilación de requerimientos, Análisis, Diseño de sistemas, Pruebas, Puesta en práctica o Instrumentación, en Reingeniería y en cualquier actividad de desarrollo que sea susceptible de ser modelada.

Cada diagrama puede ser usado con énfasis distinto en cada fase de desarrollo. Un diagrama cualquiera en una fase de análisis tendrá un énfasis lógico y mientras más se acerque al diseño y a la puesta en práctica, mayor será su énfasis físico y tecnológico.

Cabe aclarar que aunque UML es orientado a objetos preferentemente, es útil en cualquier modelo tecnológico ya que es independiente de lenguajes de programación o tecnología determinada.

Parte de las premisas en el desarrollo de UML fue que se deseaba separar los modelos de una metodología dada. En este marco, se entiende como metodología lo siguiente: "... *un conjunto de pasos ordenados con entradas y salidas previamente definidas (estas entradas y salidas bien pueden ser modelos expresados en UML u otra notación)*" [5]. UML es independiente de metodologías, por lo que puede ser utilizado (y lo es) en distintas metodologías como Fusion, Objectory, Rational Unified Process, OMT, ECM, Catalysys, etc. La independencia antes mencionada permite que las organizaciones adapten el uso de UML a la metodología que consideren más apropiada. UML es un lenguaje para hacer modelos.

Los beneficios que UML aporta son:

- Mejores tiempos totales de desarrollo (de 50 % o más).
- Mejor calidad
- Mejor soporte a la planeación y al control de proyectos.
- Mayor independencia del personal de desarrollo.
- Mayor soporte al cambio organizacional, comercial y tecnológico.
- Alta reutilización.
- Minimización de costos.

Se optó por utilizar UML para el desarrollo de este sistema. Las diversas fases en las que se aplicará son: *Análisis de requerimientos*, *Análisis*, *Diseño*, *Programación* y *Pruebas*.

3.3 Definición del problema y Análisis de requerimientos

3.3.1 Definición del problema

A continuación se describe el sistema actual:

> El sistema actual de apartados de computadoras consiste en una interfaz basada en caracteres, también conocida como interfaz en "modo texto", la cual se opera con las teclas de función, de F1 a F12, donde cada tecla realiza una actividad específica. Por ejemplo, supongamos que un usuario llega al laboratorio y quiere ocupar una máquina a partir de ese momento y durante las próximas horas. Lo primero que se realiza es escribir el login del usuario y a continuación se presiona la tecla F1 que es la tecla asociada a la entrada al laboratorio (Figura 3-6). Posteriormente se cambia la pantalla por otra en la cual se pregunta por cuánto tiempo va a utilizar el usuario una de las computadoras (Figura 3-7); presentándose un menú con algunos intervalos fijos de tiempo en los cuales se vuelven a utilizar las teclas de función (Figura 3-8). En base al tiempo de trabajo por parte del usuario, se presentan las computadoras que estarán desocupadas en ese intervalo de tiempo. El usuario observa todas las computadoras disponibles y escoge una de ellas. El recepcionista del laboratorio, entonces, escribe el número asociado a la computadora deseada y se presiona la tecla *Enter* (Figura 3-9). En ese momento la asignación es concedida y se permite la entrada del usuario.

Este sistema se desarrolló en lenguaje C utilizando el compilador de Borland versión 2.0 y es el sistema que hasta la fecha se sigue utilizando para registrar el uso y apartado de esas computadoras. Se pretende que el sistema a desarrollar, además de resolver las deficiencias del actual sistema, presente las siguientes características:

- El sistema se analizará y diseñará utilizando el paradigma orientado a objetos.
- El almacenamiento de los datos se llevará a cabo en bases de datos en lugar de realizarse en archivos "planos". Con esto, se asegura que los datos estarán seguros, consistentes, íntegros y serán de rápido acceso para obtener información útil para cada usuario.
- El sistema trabajará en ambiente de red y multiprocesos (multithreading), con lo cual varios usuarios tienen la posibilidad de apartar remota y simultáneamente las computadoras que van a utilizar.

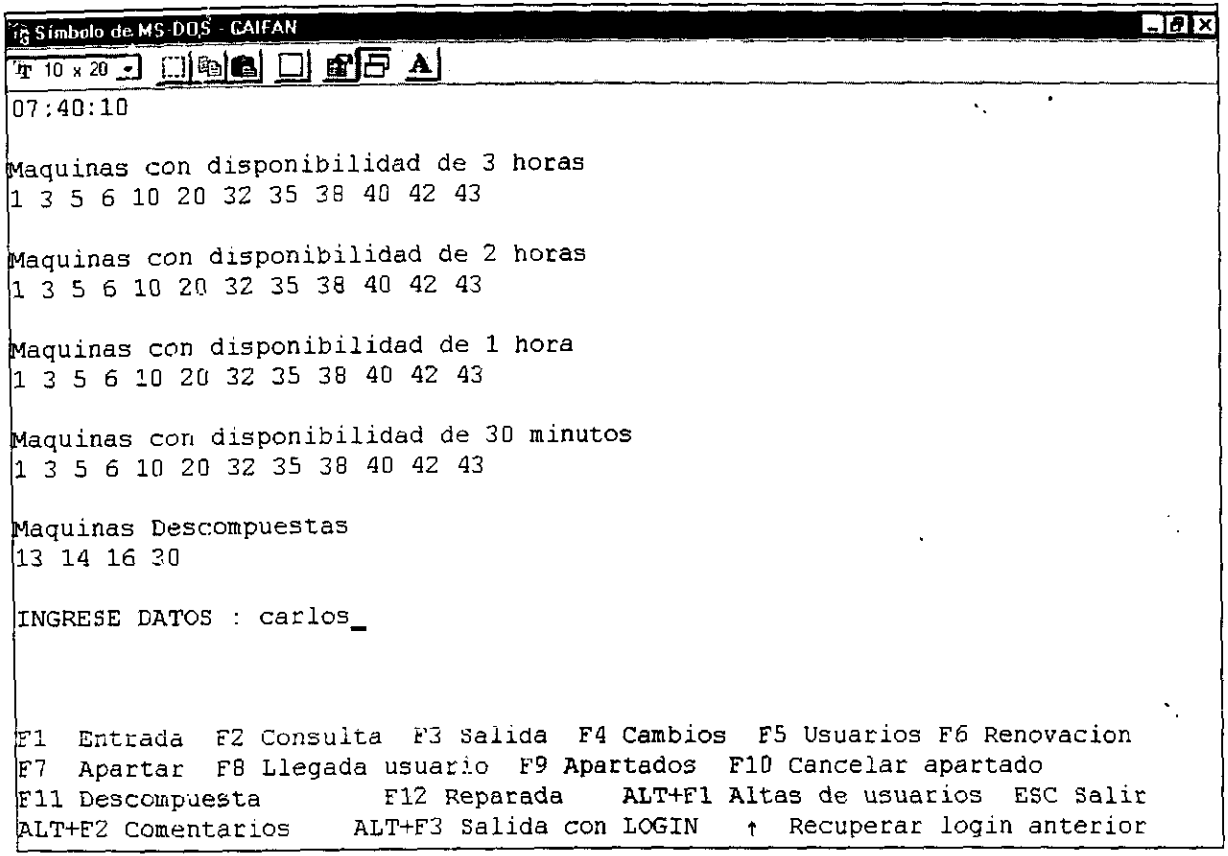

Símbolo de MS-DOS - CAIFAN
10 x 20
07:40:10
Maquinas con disponibilidad de 3 horas
1 3 5 6 10 20 32 35 38 40 42 43
Maquinas con disponibilidad de 2 horas
1 3 5 6 10 20 32 35 38 40 42 43
Maquinas con disponibilidad de 1 hora
1 3 5 6 10 20 32 35 38 40 42 43
Maquinas con disponibilidad de 30 minutos
1 3 5 6 10 20 32 35 38 40 42 43
Maquinas Descompuestas
13 14 16 30
INGRESE DATOS : carlos
F1 Entrada F2 Consulta F3 Salida F4 Cambios F5 Usuarios F6 Renovacion
F7 Apartar F8 Llegada usuario F9 Apartados F10 Cancelar apartado
F11 Descompuesta F12 Reparada ALT+F1 Altas de usuarios ESC Salir
ALT+F2 Comentarios ALT+F3 Salida con LOGIN ↑ Recuperar login anterior

Figura 3-6

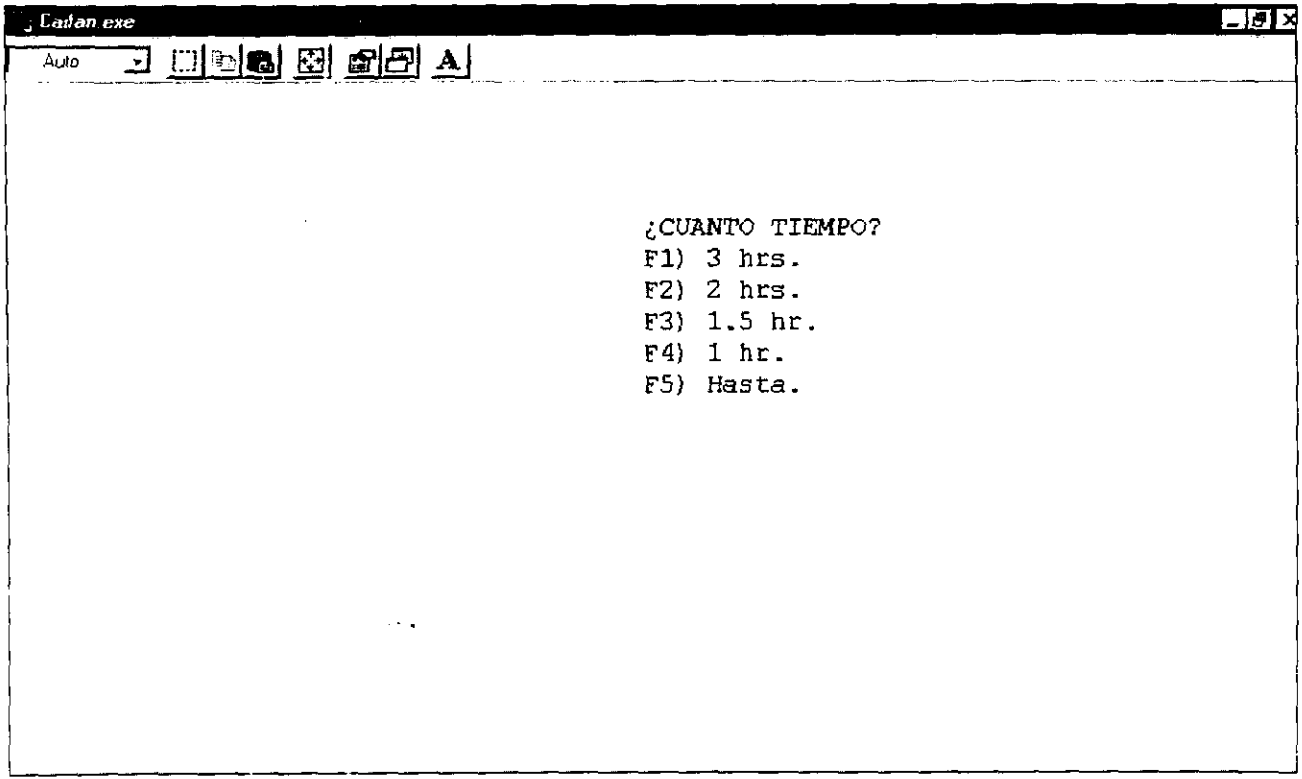

Cadan exe
Auto
¿CUANTO TIEMPO?
F1) 3 hrs.
F2) 2 hrs.
F3) 1.5 hr.
F4) 1 hr.
F5) Hasta.

Figura 3-7

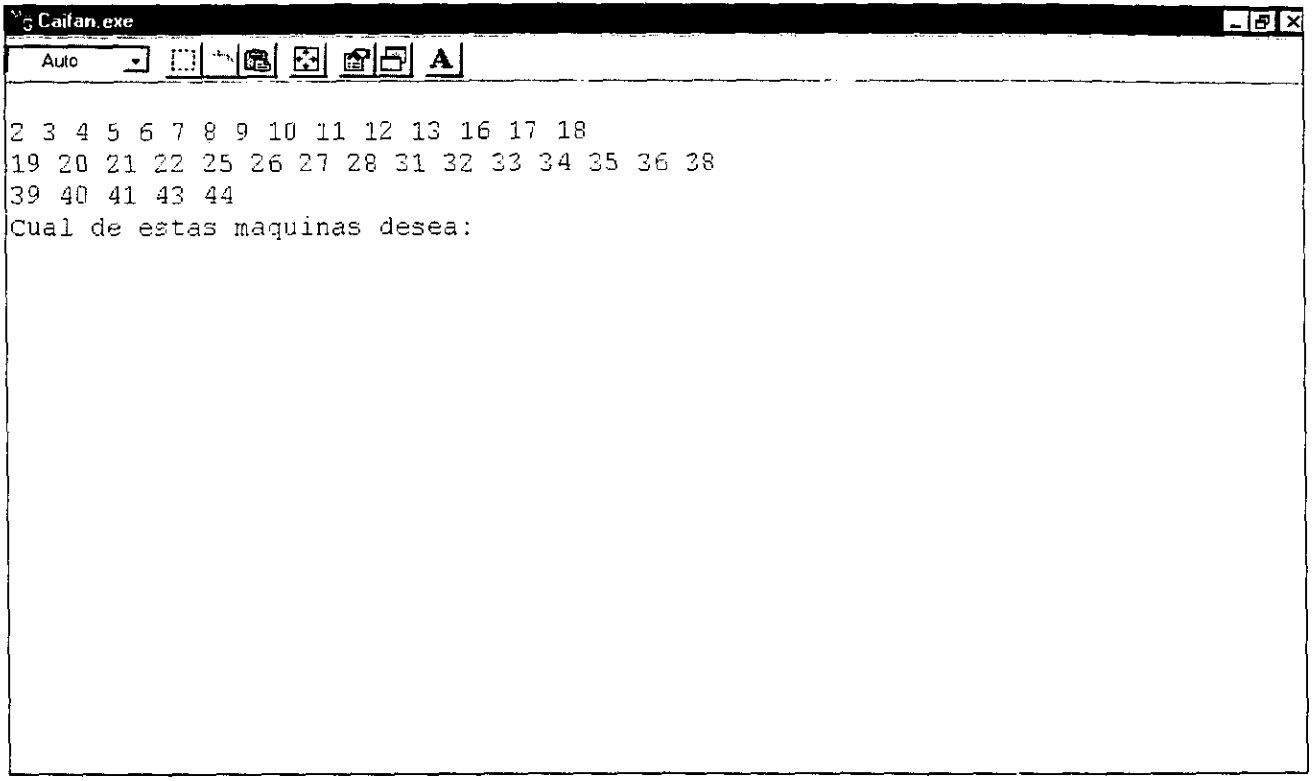

Caifan.exe
Auto
2 3 4 5 6 7 8 9 10 11 12 13 16 17 18
19 20 21 22 25 26 27 28 31 32 33 34 35 36 38
39 40 41 43 44
Cual de estas maquinas desea:

Figura 3-8

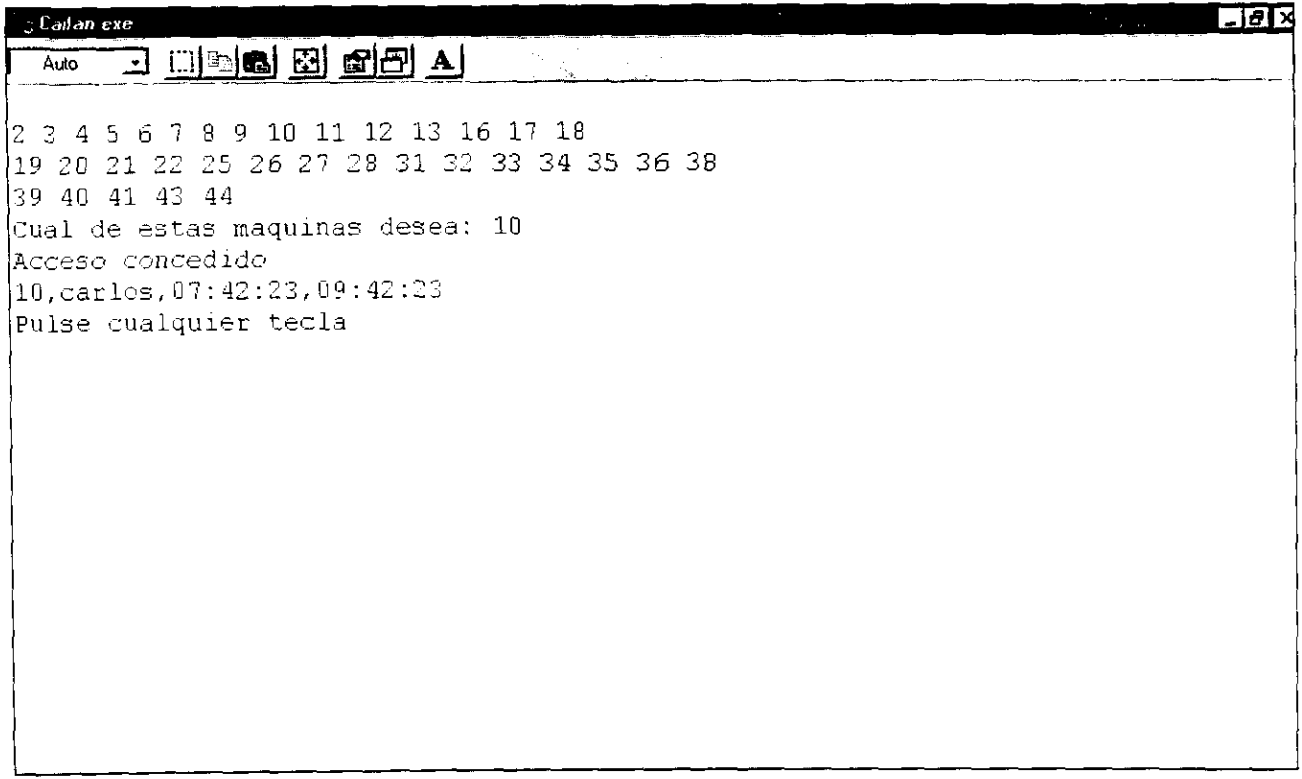

Callan exe
Auto
2 3 4 5 6 7 8 9 10 11 12 13 16 17 18
19 20 21 22 25 26 27 28 31 32 33 34 35 36 38
39 40 41 43 44
Cual de estas maquinas desea: 10
Acceso concedido
10,carlos,07:42:23,09:42:23
Pulse cualquier tecla

Figura 3-9

3.3.2 Análisis de requerimientos

Ahora se realizará el *Análisis de requerimientos*, es decir, la descripción del sistema a desarrollar y que resolverá el problema actualmente existente. Se proporcionan también prototipos o vistas del futuro sistema.

El sistema a desarrollar consiste en una interfaz basada en el World Wide Web (WWW) y presenta, inicialmente, una primera página web que permite el acceso a los servicios del sistema. Una vez que el usuario decide entrar al sistema, aparece una ventana para la validación de ese usuario que contiene dos campos de texto, uno para el login y otro para el password del usuario (Figura 3-10). Estos dos datos se le proporcionan en el laboratorio al momento que registró su alta o renovación. Si el login o el password o ambos son incorrectos, debe aparecer un cuadro de diálogo que lo indique (Figura 3-11) con un botón de *Aceptar* para aceptar la negación de entrada al sistema. Una vez que los datos proporcionados por el usuario son validados, aparece una nueva página web en la que se muestran una interfaz gráfica, con imágenes de computadoras con un número y un checkbox asignado a cada imagen. Cada computadora se muestra en un color dependiendo de su estado: azul si está disponible, rojo si está ocupada actualmente o está apartada para ocuparse posteriormente y amarillo si está fuera de servicio. Para estos dos últimos estados, el checkbox de la computadora está deshabilitado para que no pueda seleccionarse. Además, tiene 5 botones, 2 campos de texto y 2 etiquetas de texto. Las etiquetas y los campos de texto son para que el usuario escriba la hora de inicio y la hora de término del uso de la computadora.

Los 5 botones de la interfaz permiten que un usuario realice las siguientes acciones: *Consultar* la disponibilidad de computadoras tanto del momento actual como de un intervalo de tiempo especificado por el mismo usuario, *Apartar* una computadora que esté disponible en el intervalo de tiempo especificado por el mismo usuario, *Consultar el apartado previo* realizado con anterioridad por el mismo usuario, *Cancelar el apartado previo* realizado con anterioridad por el mismo usuario y *Salir* del sistema (Figura 3-12). El mapa de la sala es dinámico, es decir, si en ese momento alguna computadora fuera ocupada por otro usuario que también esté conectado al sistema, el mapa se actualiza de manera automática para el primer usuario que está observando la página web.

Un usuario solamente puede realizar un apartado al día. Las acciones no permitidas por el sistema son las siguientes:

- La falta de un dato obligatorio solicitado por el sistema como la hora de inicio, la hora de término o el número de la computadora a apartar.
- Especificar un formato inválido para la hora de inicio y/o para la hora de término. El formato de hora es de 0:00 a 24:00 horas y un ejemplo de formato inválido sería especificar la hora de inicio mayor a la hora de término, como 13:00 horas y 11:00 horas, respectivamente.
- Realizar el apartado de una computadora que ya esté reservada por otro usuario en ese mismo intervalo de tiempo.
- Realizar el apartado de una computadora si ese mismo usuario ya cuenta con un apartado previo. En este caso, el apartado anterior se conserva mientras que el nuevo apartado se cancela.

Para cada uno de los casos anteriores aparece una ventana de *Operación cancelada* con un botón de *Aceptar* que permite cerrar dicha ventana una vez aceptada la cancelación. En la Figura 3-13 se muestra, como ejemplo, el último caso en que el usuario desea realizar un apartado pero ya cuenta con un apartado previo. En tal caso, lo que puede hacer dicho usuario es cancelar su apartado previo, y en esa misma sesión, realizar el nuevo apartado. En caso contrario al cancelado, aparece una ventana de *Operación realizada con éxito* (Figura 3-14).

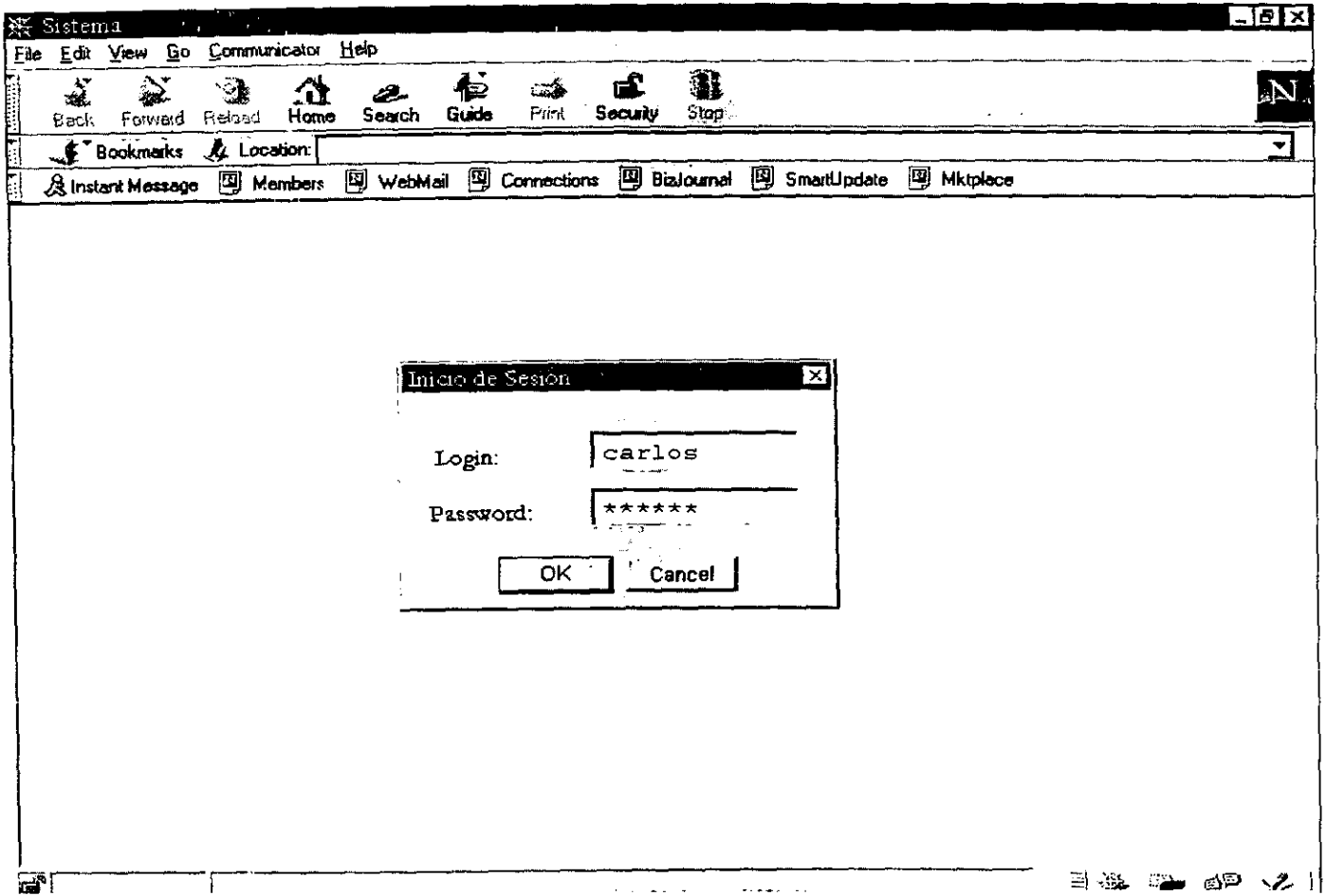

Sistema
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location:
Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace
Inicio de Sesion
Login: carlos
Password: ******
OK Cancel

Figura 3-10

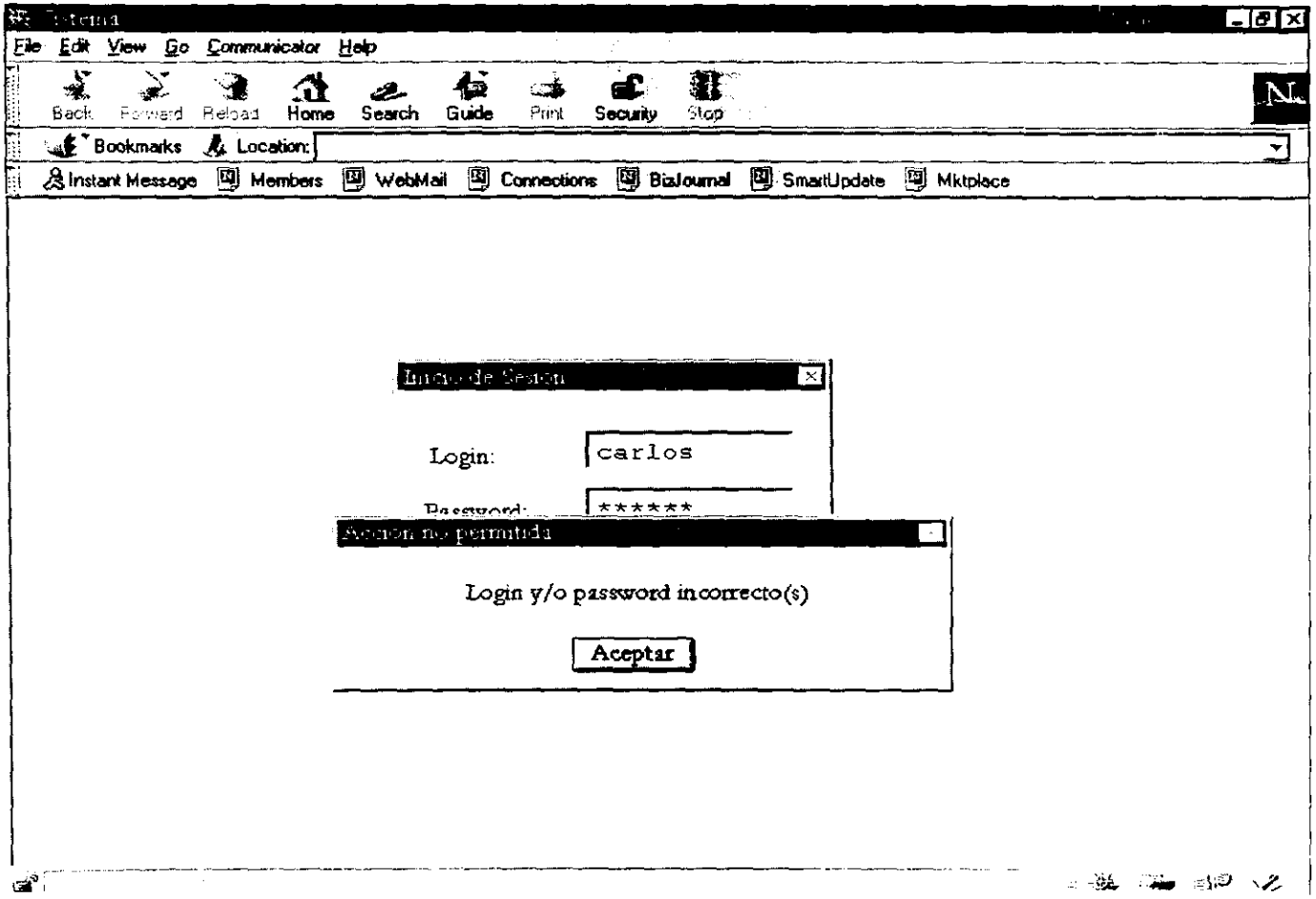

File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location:
Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace
Inicio de Sesion
Login: carlos
Password: ******
Sesion no permitida
Login y/o password incorrecto(s)
Aceptar

Figura 3-11

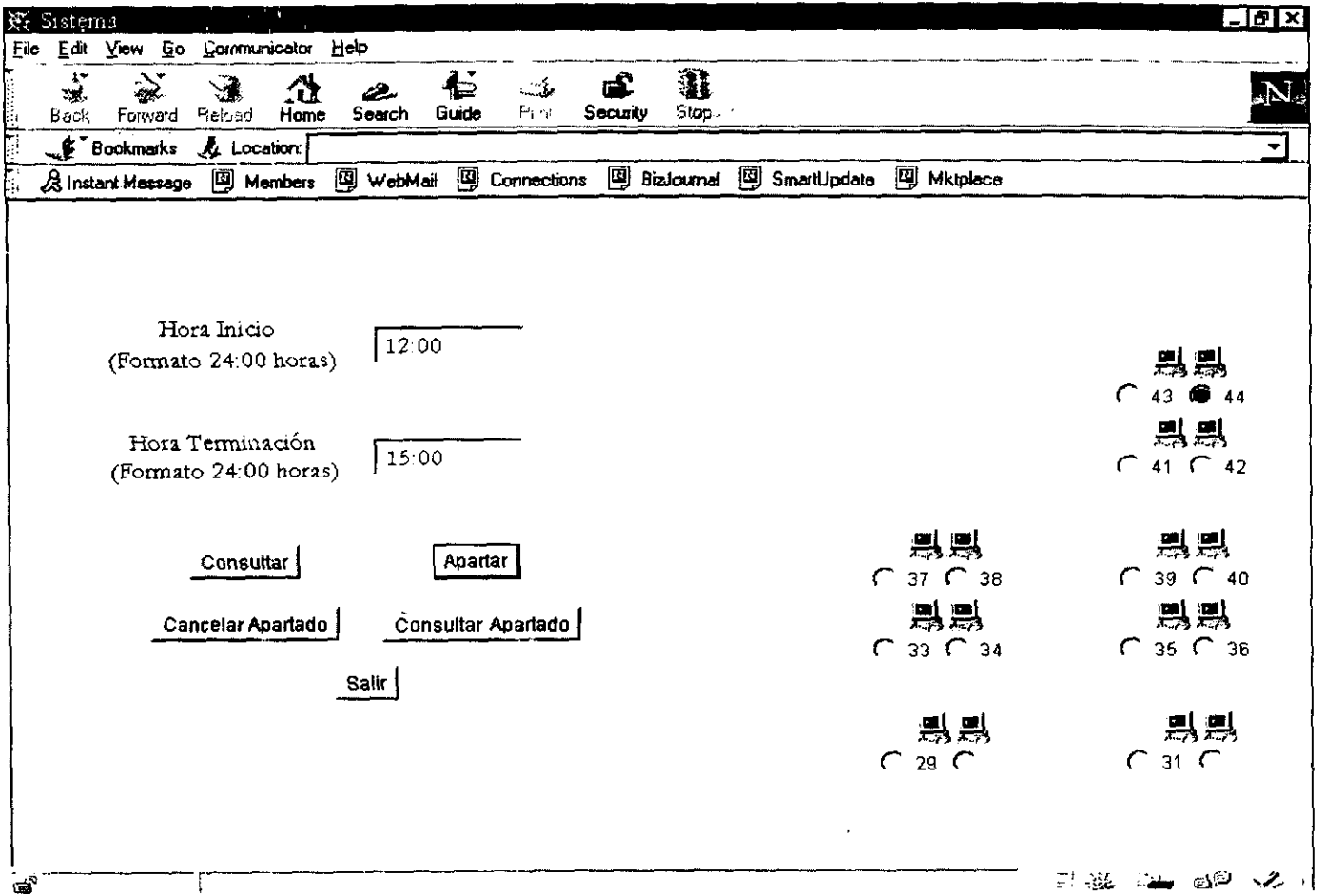

Sistema
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location:
Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace
Hora Inicio
(Formato 24:00 horas)
12:00
Hora Terminación
(Formato 24:00 horas)
15:00
Consultar
Apartar
Cancelar Apartado
Consultar Apartado
Salir
43 44
41 42
37 38
39 40
33 34
35 36
29
31

Figura 3-12

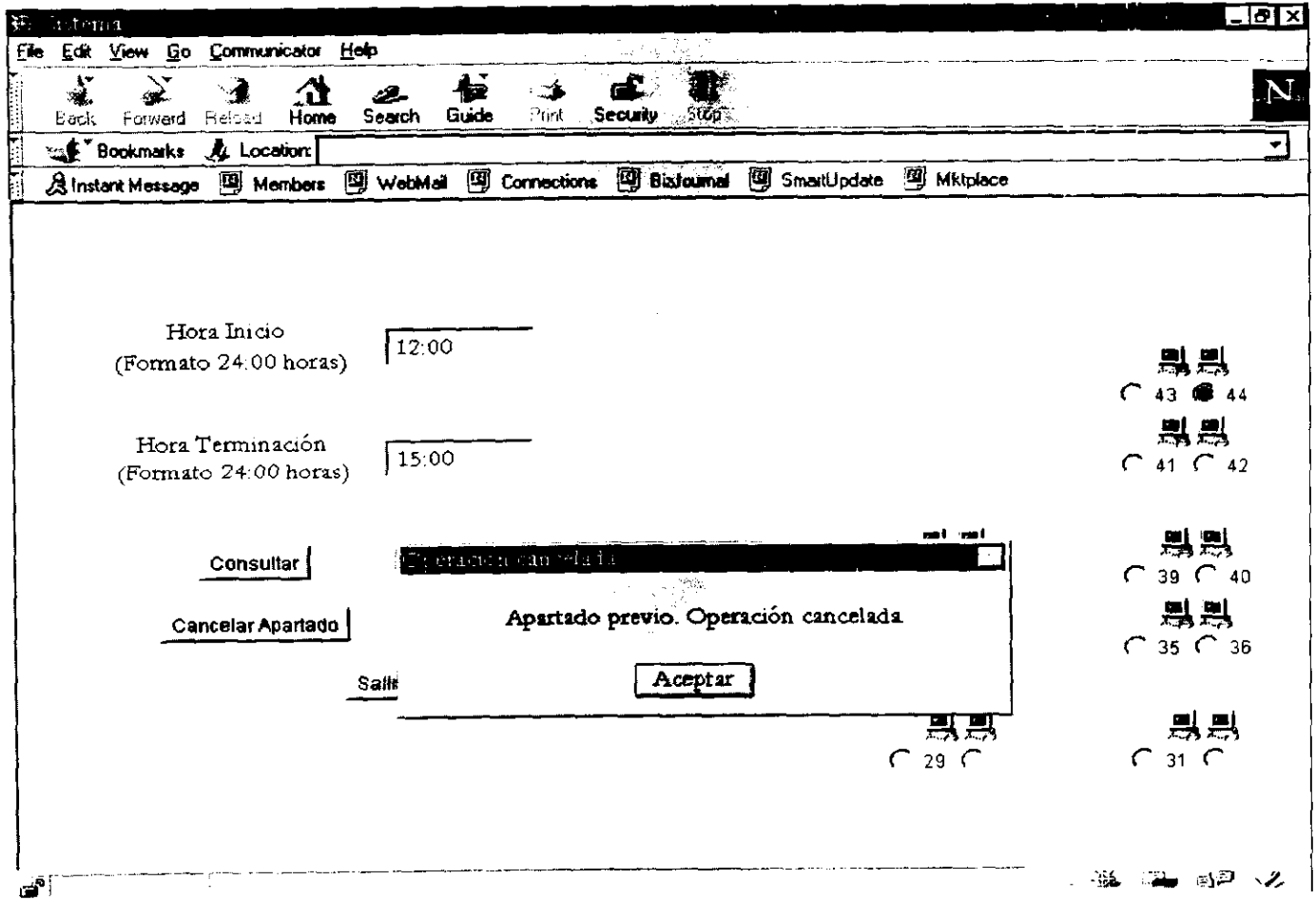

File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location:
Instant Message Members WebMail Connections Bizjournal SmartUpdate Mktplace
Hora Inicio
(Formato 24:00 horas)
12:00
Hora Terminación
(Formato 24:00 horas)
15:00
Consultar
Cancelar Apartado
Salir
Apartado previo. Operación cancelada
Aceptar
43 44
41 42
39 40
35 36
29
31

Figura 3-13

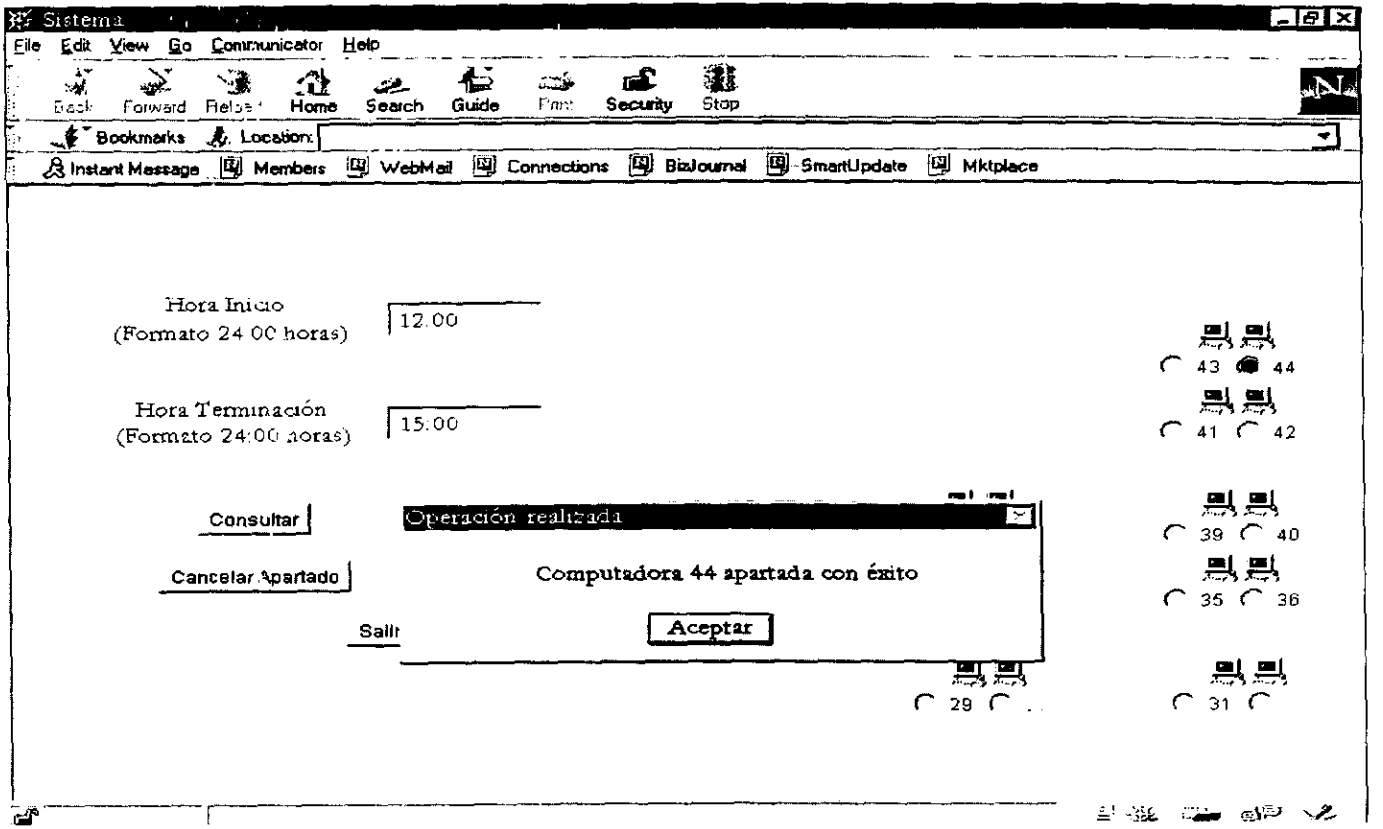

Sistema
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location:
Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace
Hora Inicio
(Formato 24 00 horas)
12.00
Hora Terminación
(Formato 24:00 horas)
15.00
Consultar
Cancelar Apartado
Salir
Operación realizada
Computadora 44 apartada con éxito
Aceptar
43 44
41 42
39 40
35 36
29
31

Figura 3-14

3.3.3 Diagrama de caso de uso ("use-case")

A continuación se realizará el ***diagrama de caso de uso o use-case*** del sistema descrito (Figura 3-15). Para estos diagramas se utilizó el software de modelado *Rational Rose 98 Enterprise* versión 4.5.8054a.

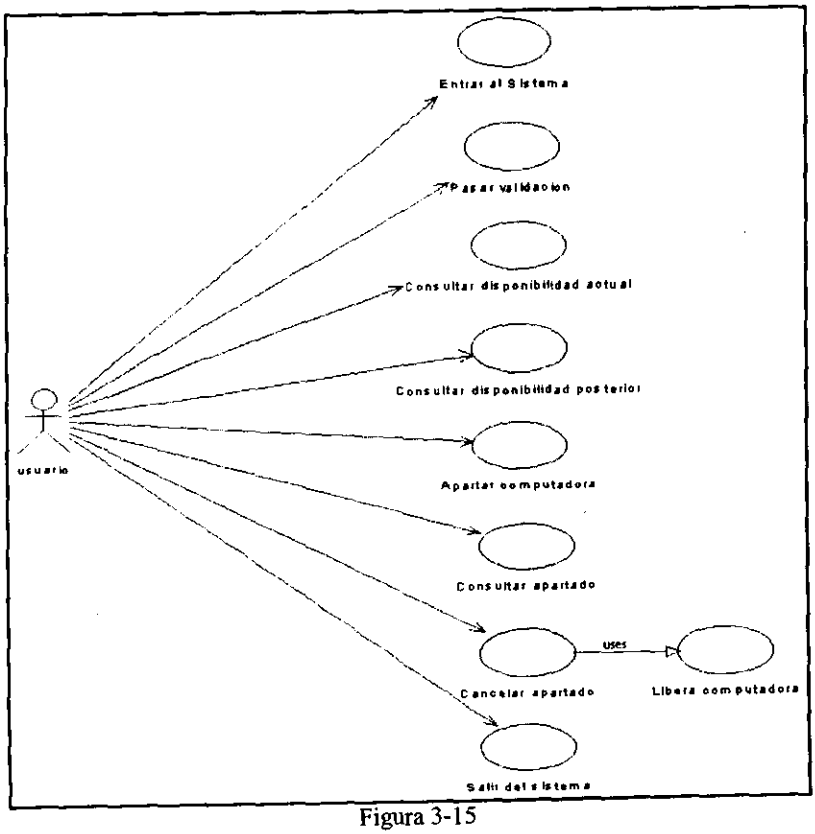


Figura 3-15

Un diagrama use-case representa la funcionalidad completa tal y como la percibe un actor y es inicializado por éste. Este diagrama no muestra clases sino acciones o funciones propias del actor, que en este caso es un usuario del laboratorio. El use-case *Cancelar apartado* se relaciona con el use-case *Libera computadora* ya que la conducta del primer use-case es utilizada por el segundo.

3.4 El Análisis

3.4.1 Diagrama de secuencia

Posteriormente, ya en la etapa de Análisis, se procedió a realizar el ***diagrama de secuencia*** (Figura 3-16) tomando como base el diagrama use-case. Este diagrama muestra la interacción de los objetos entre ellos. Es importante comentar que hasta este momento no se han considerado objetos técnicos. En UML, durante el *Análisis de los requerimientos* y el *Análisis*, no se consideran objetos técnicos que definan detalles y soluciones en el sistema de software, tales como objetos para interfaces de usuario, bases de datos, comunicaciones, etc. Todos esos objetos se consideran hasta el diseño del sistema.

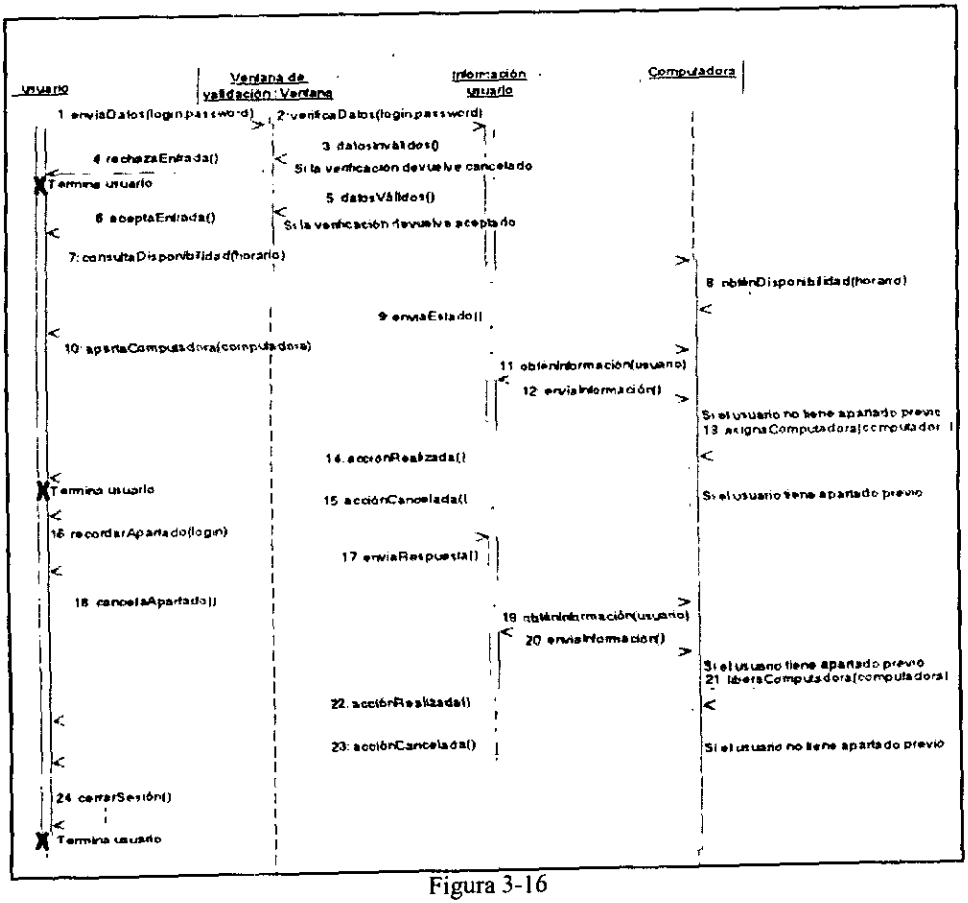


Figura 3-16

En este diagrama se observa que sólo se consideran algunos objetos y es importante aclarar que estos no serán todos los objetos a considerar dentro del sistema, ya que todavía es posible agregar nuevos objetos que no se habían considerado en el dominio del análisis así como los objetos técnicos, como se mencionó anteriormente. Los objetos considerados se representan en rectángulos con el nombre subrayado y cada uno cuenta con su línea de vida vertical que muestra la vida del objeto.

Por el momento, la interfaz inicial de la ventana de validación se considera como el objeto **Ventana de validación** y es con la cual interactúa inicialmente el actor (el objeto **usuario**). Es posible que esta ventana tenga relaciones con otros objetos tales como los botones, campos de texto, etiquetas, etc., pero posteriormente se analizará esta interfaz.

Se hace uso del objeto **Computadora** pensando en que este objeto es con el que interactuará el actor para realizar, precisamente, el apartado de una de las computadoras. **Información usuario** es un objeto que contiene, precisamente, el login, password, apartado previo, etc., de cada usuario. Este objeto, en realidad, tendrá relación con la base de datos no considerada hasta este momento.

De igual manera, los mensajes entre objetos (métodos) no se consideran de manera técnica. Se muestran con flechas horizontales en las que se indica su nombre y el paso de parámetros. Estas dos características no son definitivas, es decir, posteriormente se pueden llamar de manera diferente así como tener diferentes parámetros. Los mensajes están numerados para distinguir cuándo se ejecuta cada uno de ellos. La ejecución de un mensaje puede estar condicionada al resultado de otro mensaje y dicho resultado se representa con una etiqueta. Por ejemplo, si la verificación de login y password devuelve cancelado, entonces se ejecuta el mensaje *rechazaEntrada()* y en ese momento acaba el objeto usuario y se representa con una "**X**" en la línea de vida de ese objeto. Si la verificación devuelve una aceptación, entonces el usuario puede realizar las acciones propias del sistema. Es importante notar que el objeto Computadora envía los mensajes *asignaComputadora(computadora)* y *liberaComputadora(computadora)* a él mismo. Posteriormente en la programación, es probable, que este objeto Computadora cambie de nombre y que ese nuevo objeto tenga relación con uno de los tantos objetos Computadora que hay en el sistema y con el cual se comunicará a través de los mensajes *asignaComputadora(computadora)* y *liberaComputadora(computadora)*.

3.4.2 Diagrama de colaboración

Así mismo, se cuenta con el ***diagrama de colaboración*** (Figura 3-17), el cual se centra tanto en las interacciones y las ligas entre un conjunto de objetos colaborando entre ellos (una liga es una instancia de una asociación). Ambos, el diagrama de secuencia y el diagrama de colaboración, muestran interacciones, pero el diagrama de secuencia se centra en el tiempo mientras que el diagrama de colaboración se centra en el espacio. Las ligas muestran los objetos actuales y cómo ellos se relacionan unos con otros. Así como los diagramas de secuencia, los diagramas de colaboración pueden ser utilizados para ilustrar la ejecución de una operación, una ejecución de un use-case o simplemente un escenario de interacción dentro del sistema. En este diagrama también se representa a los objetos en cajas rectangulares y con el nombre subrayado. Las ligas se dibujan con líneas y se puede agregar una etiqueta para un mensaje y un número que define la secuencia de las ligas.

En el software de modelado *Rational Rose*, a partir del diagrama de secuencia, se genera de manera automática el diagrama de colaboración, como se muestra a continuación.

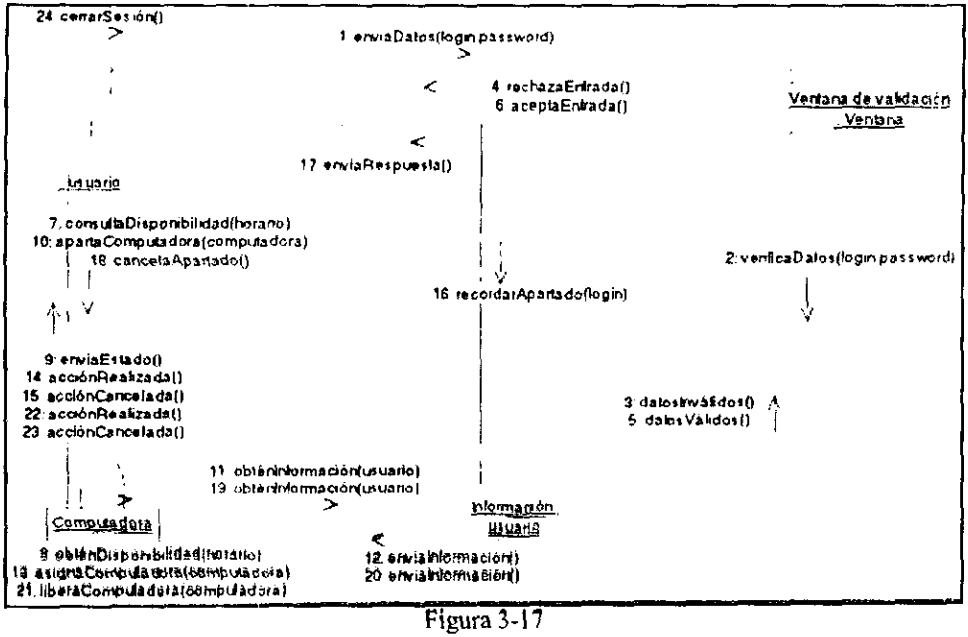


Figura 3-17

3.4.3 Diagrama de clases

A continuación se realizará el ***diagrama de clases*** tomando como base los diagramas de secuencia y de colaboración por lo que se manejarán los objetos que ahí se consideraron pero ahora a nivel de clases. Además, se pueden agregar nuevas clases que no se habían considerado y este paso deberá ser realizado por expertos en el dominio del problema. Para poder definir las clases, UML sugiere seis características selectivas que debe utilizar el analista para considerar una clase candidato en el modelo de análisis:

1. *Información retenida*. La clase será útil durante el análisis sólo si la información sobre el mismo ha de ser almacenada, transformada, analizada o manejada en algún otro modo. La información puede referirse a conceptos que deberán estar siempre registrados en el sistema, eventos o transacciones que ocurren en un momento específico.
2. *Sistema externo*. Si se tiene un sistema externo a este sistema, entonces es de interés en la etapa de modelado. Los sistemas externos deberán ser vistos como clases que el sistema contendrá o con los cuales interactuará.
3. *Patrones, librerías de clases o componentes*. Si se tienen patrones, librerías de clases o componentes, generalmente éstos son clases candidatos.
4. *Dispositivos que el sistema maneja*. Dispositivos técnicos que maneja el sistema se convertirán en clases que manejarán esos dispositivos.
5. *Partes organizacionales*. Especialmente en modelos de negocio, todas las partes que representan a la organización, serán clases candidatos.
6. *Roles de actores*. Los roles de actores serán vistos como clases, por ejemplo, usuario, operador del sistema, administrador, cliente, etc.

Basándonos en estas características anteriores, se seleccionan las clases reales de este sistema y obtenemos la figura siguiente (Figura 3-18).

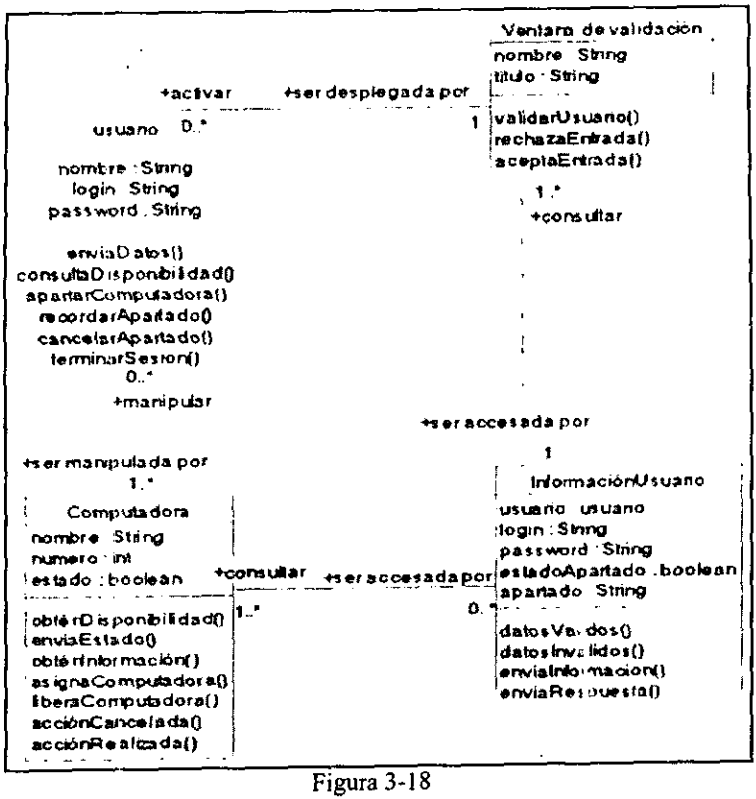


Figura 3-18

Por el momento se van a considerar las clases cuyos objetos se consideraron en el diagrama de secuencia y en el diagrama de colaboración. En el diagrama de clases se considera la frase de la relación con una etiqueta en la línea de la relación y del lado de la clase que va a aplicar esa relación. Así mismo, se considerará la cardinalidad de la relación entre dos clases, tomando la unidad en la clase de origen y la cardinalidad respectiva en la clase destino. Por ejemplo, en el diagrama de clases se observa que la clase **usuario** y la clase **Ventana de validación** tienen relación entre ellas. La relación se lee: "*un usuario puede* ***activar*** *una ventana de validación y una ventana de validación puede* ***ser desplegada por*** *cero o más usuarios*".

Cuando se presenta un diagrama de clases, hay que observar que es muy común que se presente una relación muchos a muchos entre dos clases. Por ejemplo entre la clase **usuario** y entre la clase **Computadora**: "*un usuario puede* ***manipular*** *una o más computadoras y una computadora puede* ***ser manipulada por*** *cero o más usuarios*" así como entre las

clases **Computadora** e **InformaciónUsuario**. Cuando esto sucede, lo más recomendable es transformar esa relación en dos relaciones uno a muchos. En base a lo anterior, podemos crear una clase intermedia entre usuario y Computadora a la que llamaremos **Mapa**, en alusión al mapa que contiene las computadoras del laboratorio. De igual manera, podemos crear una clase entre Computadora e InformaciónUsuario pero podemos utilizar la misma clase intermedia Mapa. El diagrama de clases quedará como el mostrado en la figura 3-19.

La nueva clase Mapa conserva las relaciones de la anterior clase Computadora y ésta se relaciona, ahora, con la nueva clase Mapa en una relación que no debe ser muchos a muchos. En este caso, la nueva relación es uno a muchos: "*un mapa puede* ***contener*** *una o más computadoras y una computadora puede* ***estar contenida*** *en sólo un mapa*". Al relacionarse de esta manera, la clase Mapa es la que va a contener todos los objetos Computadora y además va a contener como atributo la fecha actual: hora y día que son muy importantes en este sistema ya que en la información de cada usuario cuando un usuario realice un apartado, éste se debe agregar con la fecha en que se realizó la operación por parte de ese usuario. Con esto, se da la posibilidad de realizar alguna operación dentro del sistema, como la consulta y el apartado de una computadora, con varios días de anticipación. Así mismo, los métodos que anteriormente contenía la clase Computadora ahora son ejecutados por la clase Mapa ya que esos métodos realizaban el envío de mensajes a las clases usuario e InformaciónUsuario y la clase Computadora ahora sólo contiene dos métodos: *asignaEstado(numComputadora)* y *obténEstado(numComputadora)*. Esto se debe a que lo realizado entre las clases Mapa, usuario e InformaciónUsuario repercutirá en que una computadora cambie a un estado nuevo o envíe como respuesta su estado actual: libre, ocupada o fuera de servicio.

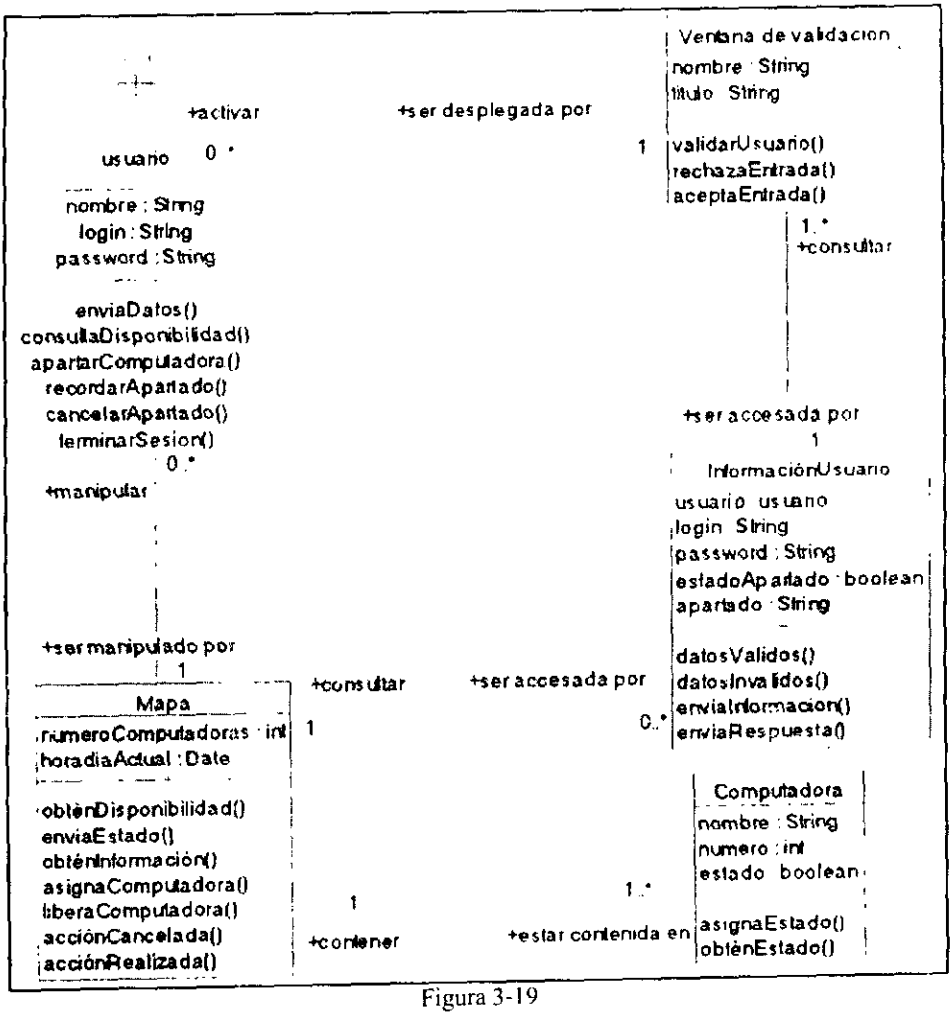

+activar
+ser desplegada por
Ventana de validacion
nombre : String
titulo : String
usuario
0 *
1
validarUsuario()
rechazaEntrada()
aceptaEntrada()
nombre : String
login : String
password : String
1..*
+consultar
enviaDatos()
consultaDisponibilidad()
apartarComputadora()
recordarApartado()
cancelarApartado()
terminarSesion()
+ser accesada por
1
0..*
InformaciónUsuario
+manipular
usuario usuario
login : String
password : String
estadoApartado : boolean
apartado : String
datosValidos()
datosInvalidos()
enviaInformacion()
enviaRespuesta()
+ser manipulado por
1
+consultar
+ser accesada por
Mapa
numeroComputadoras : int
horadiaActual : Date
1
0..*
Computadora
nombre : String
numero : int
estado : boolean
obtènDisponibilidad()
enviaEstado()
obtèninformación()
asignaComputadora()
liberaComputadora()
acciónCancelada()
acciónRealizada()
1
1..*
+contener
+estar contenida en
asignaEstado()
obtènEstado()

Figura 3-19

3.4.4 Diagrama de estados

Posteriormente se debe realizar el ***diagrama de estados*** el cual captura el ciclo de vida de los objetos, subsistemas y sistemas. Dicho diagrama determina los estados que un objeto puede tener y cómo los eventos afectan esos estados a través del tiempo. Un diagrama de estado debe abarcar todas las clases que tengan estados y conducta definidos claramente.

Todos los objetos tienen un estado y éste es el resultado de actividades previas ejecutadas por el objeto. Ese estado está determinado por los valores de los atributos de este objeto y sus relaciones con otros objetos. Una clase puede tener un atributo que especifique el estado, o el estado puede ser determinado por los valores de los atributos "normales" del objeto.

Por lo anterior, se determinó que la clase **Computadora** tiene un atributo que varía con el tiempo y el cual es afectado por la relación de la clase Computadora con las clases restantes: el estado. En consecuencia, un objeto de la clase Computadora cuenta con el atributo **estado** y en base a éste se realizará el diagrama de estados del objeto Computadora (Figura 3-20). En este diagrama no se consideran ni clases ni objetos, sólo estados (rectángulos con las esquinas redondeadas) y transiciones de estados (líneas entre los estados). Además este diagrama cuenta con un estado inicial (un círculo relleno) y un estado final (una circunferencia con el centro relleno). En este análisis estamos partiendo del estado inicial Computadora libre o desocupada y contando con dos estados más: Computadora reservada y Computadora fuera de servicio. Las etiquetas en las transiciones de estados, cuentan con una sintaxis específica. En primer lugar se determina el evento, que realiza el cambio de un estado a otro, junto con sus parámetros; en segundo lugar, de manera opcional y entre corchetes ([ ]), se especifica una condición para que se realice el cambio de estado; después se especifica una línea diagonal (/) y por último, una acción que será el resultado del cambio de estado como, por ejemplo, el incremento o decremento de una variable.

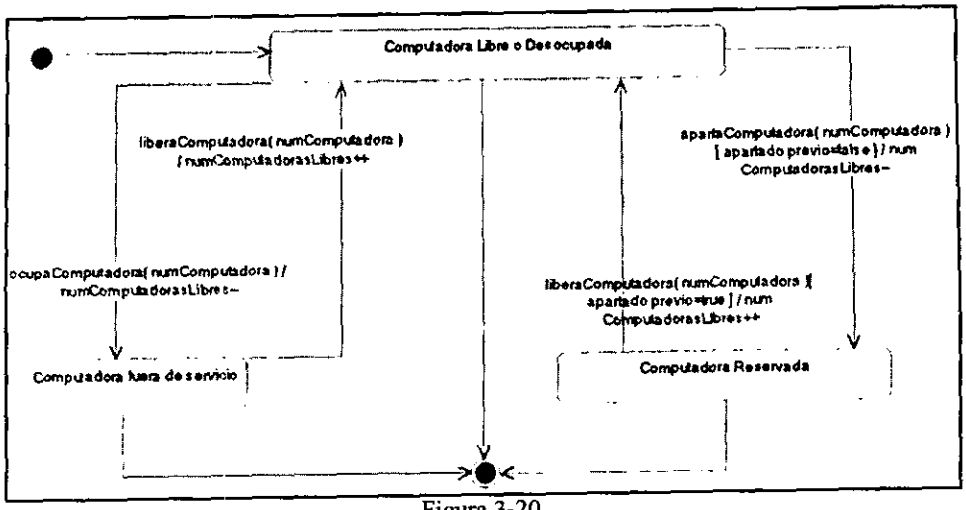


Figura 3-20

Con lo anterior concluye el Análisis y pasamos a la etapa de Diseño.

3.5 El Diseño

En el Diseño se expande el análisis agregando clases técnicas como las interfaces gráficas de usuario (GUI), manejo de objetos para almacenar datos en las bases de datos, comunicación con otros sistemas externos, etc. El Diseño es el "puente" entre el dominio del problema y la puesta en práctica técnica.

3.5.1 Diagrama de componentes

Dentro de esta etapa se crea el diagrama de componentes que describe componentes de software y sus dependencias con otros componentes, representando la estructura del código. Los componentes de software pueden ser: componentes de código, componentes binarios que son los generados por la compilación de los componentes de código y los componentes ejecutables.

Es importante señalar que en el diseño ya se puede haber seleccionado el lenguaje en el cual se va a programar el sistema pero no es obligatorio. De hecho, en este sistema todavía no se contempla el lenguaje a utilizar, por lo que el diagrama de componentes no contempla componentes de software específicos, como lo muestra la figura 3-21.

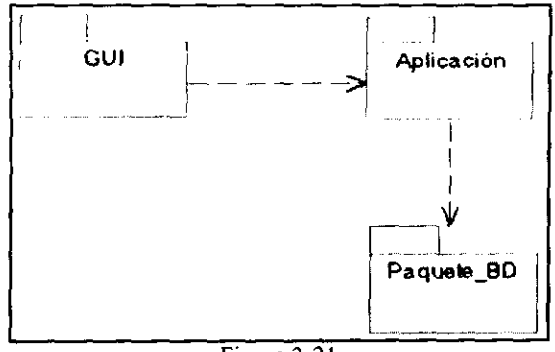


Figura 3-21

En este diagrama se pueden manejar *paquetes*, que son contenedores de clases utilizados para mantener el espacio de nombres de clases dividido en compartimentos, de manera que se utilizan para representar subsistemas del sistema en el mundo físico. Es decir, dos paquetes pueden tener una clase Ventana cada uno de ellos, evitando un conflicto de nombres. Los paquetes se almacenan de manera jerárquica y físicamente son subdirectorios del sistema de archivos. En el diagrama, se representan con folders y como se puede observar, en este momento se están manejando tres paquetes: la Interfaz Gráfica de Usuario (GUI, Graphic User Interface), la Aplicación y el paquete de Bases de Datos.

Revisando el análisis de requerimientos, se observa que el usuario entra al sistema a través de una ventana de validación y, ya una vez dentro, aparece un mapa de computadoras. Toda esa interfaz se representa en el primer paquete. Debido a que la conexión de los usuarios es a través de navegadores de Web, entonces se debe considerar un servidor de Web que atienda a los clientes (usuarios). En este servidor de Web debemos considerar una aplicación de software, la cual corresponde al segundo paquete, que reciba la información que envían los usuarios del laboratorio y la envía a la base de datos, que está representada en un tercer paquete.

Cada paquete se liga con otros a través de *dependencias*, que se representan con flechas de líneas discontinuas que van del componente dependiente al componente del cual depende. El paquete GUI depende del paquete Aplicación ya que a éste le envía las peticiones de los usuarios. El paquete Aplicación depende del paquete de Bases de Datos ya que a éste envía toda la información de los usuarios y dependiendo de lo que éste paquete le envíe de respuesta, comunicará a los usuarios que su operación se realizó con éxito o se canceló.

Cada paquete debe tener un diagrama de componentes para representar las clases que contiene internamente, similar a hacer un acercamiento o "zoom" al interior de cada uno de los paquetes. Por lo tanto, el diagrama de componentes del paquete GUI se muestra a continuación en la figura 3-22.

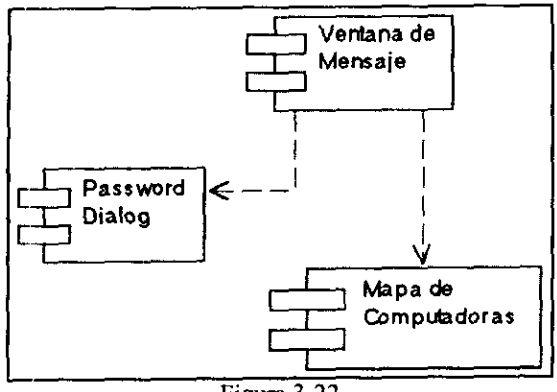


Figura 3-22

Los *componentes de software* se representan con un rectángulo que contiene dos pequeños rectángulos en su extremo izquierdo. En el paquete GUI se están considerando tres componentes: PasswordDialog que es la ventana que solicita el login y el password de cada usuario, el mapa de computadoras del laboratorio y la ventana de mensaje que depende de los otros dos componentes de software anteriores ya que, dependiendo de lo que realice el usuario tanto en la clase PasswordDialog como en la clase Mapa, aparecerá una ventana de mensaje con la respuesta a la acción realizada. Al nivel de programación, podemos decir que en estas dos clases se crean objetos de la clase Ventana.

En el paquete Aplicación también se realiza un diagrama de componentes, como se muestra en la figura 3-23. Hasta este momento sólo se ha contemplado, precisamente, la clase Aplicación que va a recibir las peticiones de los usuarios y que carece de interfaz gráfica.

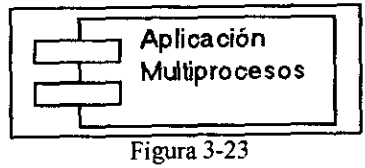


Figura 3-23

En el caso del paquete de Bases de Datos, hasta el momento, éste no cuenta con componentes de software sino con tablas que contienen los datos de los usuarios, por lo que más adelante se explicará el contenido de ese paquete.

3.5.2 Diagrama de despliegue

Por último, se realiza el diagrama de despliegue, el cual contiene los nodos y las conexiones que muestran la arquitectura del sistema en tiempo de ejecución a través de procesadores, dispositivos y los componentes de software que se ejecutan en esta arquitectura. Esta es la última descripción física de la topología del sistema, describiendo la estructura de las unidades de hardware y el software que se ejecuta en cada unidad, como se muestra en la figura 3-24.

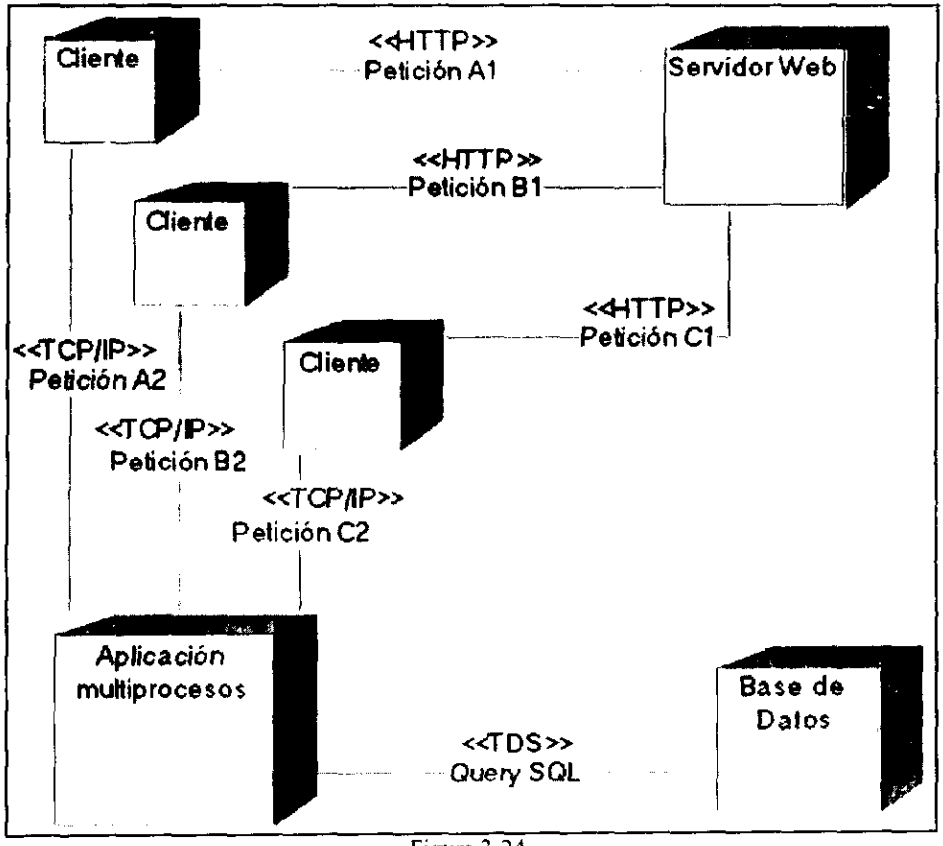


Figura 3-24

Los nodos se representan con cubos en tres dimensiones con su nombre en el interior. Si el nodo representa a una instancia en lugar de una clase, el nombre va subrayado. Las conexiones se representan con líneas discontinuas y contienen el nombre y el estereotipo de la conexión. El nombre es el identificador de la misma y el estereotipo indica el protocolo de comunicaciones entre los dos nodos implicados.

En este caso, se han colocado tres nodos **Clientes** para indicar que el sistema es multiusuarios. Los clientes se comunican primero con el nodo **Servidor Web** a través de las conexiones *Petición A1*, *Petición B1* y *Petición C1* con el protocolo *HTTP* (*Hypertext Transfer Protocol*) que es el protocolo utilizado en el WWW (World Wide Web). Una vez que el servidor web contestó sus peticiones, ahora los clientes se comunican con el nodo **Aplicación multiprocesos** que va a recibir la información de cada usuario. Esta comunicación se realiza a través de las conexiones *Petición A2*, *Petición B2* y *Petición C2* con los protocolos Internet: *TCP/IP* (*Transfer Control Protocol* e *Internet Protocol*). Posteriormente, este nodo se comunica con el nodo **Base de datos** a través de la conexión *Query SQL* por medio del protocolo *TDS* (*Tabular Data Stream*) que es el protocolo que utiliza cada uno de los clientes para comunicarse con un Sistema Manejador de Bases de Datos (DBMS, Database Management System) Sybase, que es el sistema con el que se cuenta actualmente en la División de Ingeniería Electrica.

Una vez concluido tanto el análisis como el diseño, se procede a realizar la programación o desarrollo del sistema. Debido a que el análisis y el diseño se realizaron en base al Paradigma Orientado a Objetos, la programación del sistema se inclinaba a favor de este paradigma ya que, una vez realizados el análisis y el diseño orientados a objetos, no se recomienda utilizar un lenguaje procedural estructurado. Por lo tanto, la fase de programación se realizará en un *Lenguaje Orientado a Objetos* (*OOL, Object Oriented Language*).

A continuación se analizarán algunos de los lenguajes de programación orientados a objetos con más aceptación por parte de los programadores, así como los principales sistemas manejadores de bases de datos que actualmente predominan en la industria.

4. Herramientas de Desarrollo

Para el desarrollo de un sistema que sea la solución del problema planteado, primero se optó por analizar las posibles herramientas de software con que se cuenta (lenguajes de programación, ambientes de desarrollo, manejadores de bases de datos, etc.) y desarrollar, a partir de éstas, un sistema integral que solucione los problemas y satisfaga las necesidades de trabajo.

Además, si consideramos el hecho de trabajar un sistema manejador de bases de datos en un ambiente de red, con protocolos TCP/IP que nos permitan conectividad desde, prácticamente, cualquier punto del planeta, entonces estamos ante la necesidad de un sistema que sea abierto e independiente de la plataforma desde la cual se desea ejecutar la aplicación. Un lenguaje que reúne estas características es, por ejemplo, el lenguaje de programación Java.

Considerando realizar el sistema en Java tiene sus ventajas y, por qué omitirlo, sus desventajas. Debido a la naturaleza de este lenguaje, el alcance del sistema será el poder ejecutarse desde la red mundial Internet. Es decir, los alumnos van a tener la posibilidad de apartar las computadoras desde lugares remotos a través del Web. Es por ello que nuestro análisis gira en torno a este lenguaje y al *DBMS* (*Database Management System, Sistema Manejador de Bases de Datos*) de Sybase Inc.

4.1 El lenguaje de programación Java

Java es sólo una parte del conjunto integrado de sistemas que brinda soporte a la comunicación en el Web. El nombre de Java se refiere al lenguaje de programación creado por Sun Microsystems que se utiliza para crear contenido ejecutable que puede distribuirse a través de redes. Utilizado de manera genérica, el nombre de Java se refiere a un grupo de herramientas de software que permite crear e implantar contenido ejecutable utilizando el lenguaje de programación Java.

Por lo anterior, podemos afirmar que Java es:

- un lenguaje de programación, y
- una plataforma.

La definición "formal" del lenguaje de programación Java, proporcionada por Sun Microsystems, es la siguiente:

> *"Java es un lenguaje de programación de alto nivel, simple, orientado a objetos, distribuido, interpretado, sólido, seguro, de arquitectura neutral, portable, de alto desempeño, multihilos y dinámico"* [6].

Comparado con otros lenguajes como C y C++, Java es prácticamente nuevo. La invención de Java empezó en Sun Microsystems de California al mismo tiempo que se gestaba World Wide Web, en Suiza, en 1991. La meta de este primer equipo de desarrollo, incluyendo al creador de Java, James Gosling, era generar productos electrónicos para el consumidor que pudieran ser sencillos y estar libres de errores. Lo que se necesitaba era una manera de crear código independiente de la plataforma y, que por lo tanto, permitiera que el software se ejecutara en cualquier Unidad Central de Procesamiento (CPU).

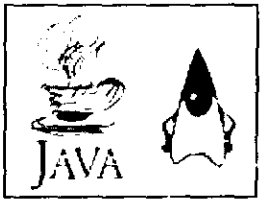
Como punto de partida para crear un lenguaje de computación que implantara esta independencia de la plataforma, el equipo de desarrollo se concentró en C++. Sin embargo, el equipo no pudo lograr que C++ hiciera todo lo que querían para crear un sistema que ofreciera apoyo a una red distribuida de dispositivos heterogéneos de comunicación. El equipo abandonó el lenguaje C++ e ideó un lenguaje llamado Oak (que después se rebautizó como Java). Hacia el otoño de 1992, el equipo había creado un proyecto llamado Star 7, que era un control remoto personal y manual.

El equipo de desarrollo fue incorporado como FirstPerson, Inc., pero entonces perdieron un concurso para producir un equipo de televisión de alta calidad para Time-Warner. A mediados de 1994, el crecimiento de la popularidad del Web atrajo la atención del equipo. Decidieron que podían construir un excelente visualizador mediante la tecnología Java. Con el objetivo de llevar al Web su sistema de programación en tiempo real e independiente del CPU, construyeron un visualizador de Web.

Ese visualizador, llamado WebRunner, se programó utilizando Java y se completó a principios del otoño de 1994. Los ejecutivos de Sun Microsystems quedaron impresionados y vieron las posibilidades tecnológicas y comerciales que podían derivarse del nuevo visualizador: herramientas, servidores y ambientes de desarrollo.

El 23 de mayo de 1995, Sun Microsystems, Inc. presentó formalmente a Java™ y a Hot Java™, nombre posterior al visualizador WebRunner, en la Sun World '95 de San Francisco

(Figura 4-1). La asociación de Netscape y Sun Microsystems llevó la tecnología Java a los visualizadores de Netscape a finales de 1995.



Logo del lenguaje de programación Java™ y Duke™, la mascota del lenguaje Java
Figura 4-1

4.1.1 Limitaciones del lenguaje Java frente a otros lenguajes de programación

A pesar de ser revolucionario, Java cuenta tanto con ventajas como con desventajas que deben estar presentes a la hora de la elección del lenguaje de programación.

A continuación se muestra una tabla comparativa entre el lenguaje de programación Java y otros posibles lenguajes de programación a utilizar para el desarrollo de este sistema (Figura 4-2).

Lenguaje/ Característica	C++	Java	SmallTalk	XML
Orientado a Objetos	Híbrido	Está discutida su naturaleza puramente OO	Puramente OO	No completamente, ya que es un lenguaje de etiquetas (tags).
Multiplataforma	No, excepto ANSI C++	Multiplataforma	Multiplataforma	Esta característica es propia de los navegadores que soportan XML
Experiencia previa para el desarrollo de este proyecto	Poca experiencia	2 años aproximadamente	Nula	Nula
Claridad y Facilidad de uso	Sintaxis compleja	Sintaxis un poco menos compleja que C++	Diversas opiniones	Sintaxis sencilla
Generación de Código	Generación rápida de archivos ejecutables	Generación un poco lenta de archivos interpretados	Generación (no tan rápida como C) de archivos interpretados	Generación rápida de archivos fuentes scripts que son interpretados por el navegador
Tiempo de compilación y Eficiencia del código generado	Excelente	Bueno	Bueno	Excelente tiempo de ejecución (Los archivos XML no se compilan)
Manejo de memoria	El usuario puede manejarla a su gusto y no cuenta con recolector de basura	Cuenta con recolector de basura	Cuenta con recolector de basura	No maneja la memoria del host cliente
Caídas del sistema	Todo el tiempo	Rara vez	Rara vez	Rara vez
Desarrollo de GUI	No estándar y es dependiente de la plataforma	Todavía es compleja pero es multiplataforma	Muy buena y fácil de usar	Muy buena
Conectividad a bases de datos	Completa	Completa	Completa	Completa
Soporte al Web en los navegadores más populares	Actualmente ya cuenta con soporte	Desde sus inicios	Actualmente ya cuenta con soporte	Desde sus inicios pero se necesita de un navegador que soporte XML y actualmente no son muy populares.

Figura 4-2

4.1.2 Características y alcances del lenguaje Java.

La tecnología de Java puede desplegarse en sistemas incrustados (como en aparatos electrónicos de consumo: dispositivos manuales, teléfonos y videocaseteras). Mitsubishi Electronics viene trabajando para utilizar tecnología de Java en esos dispositivos.

Java permite que los programadores creen contenidos que puedan distribuirse a los usuarios y que éstos puedan ejecutar en sus computadoras. Este software es capaz de proporcionar soporte a cualquier cosa que el programador pueda imaginar: hojas de cálculo, procesadores de textos, animaciones y juegos interactivos. Con la página de Web como plataforma de distribución, este software puede ofrecer soporte a diferentes tareas de información con una verdadera interactividad; los usuarios pueden obtener retroalimentación continua e instantánea para aplicaciones en la visualización, la animación y el cálculo.

Un programador de Java puede crear:

- **applets**: Programas elementales incluidos en páginas de HTML a través de la etiqueta <APPLET> </APPLET> y que se despliega en un visualizador que soporte Java.

- **aplicaciones**: Programas escritos en Java que se ejecutan independientemente del visualizador del usuario y en *modo monousuario (stand alone)*. Esto se realiza utilizando el intérprete de Java, `java`.

- **manipuladores de protocolo**: Programas que se cargan en un visualizador que soporta Java y que interpretan un protocolo. Estos protocolos incluyen estándares como HTTP o protocolos definidos por el usuario.

- **manipuladores de contenido**: Un programa cargado en un visualizador que soporta Java, que interpreta los archivos de un tipo definido por el programador de Java. Éste proporciona el código necesario para que un visualizador que acepte Java despliegue e interprete este formato especial.

- **servlets**: Programas módulos que extienden las funcionalidades de los servidores orientados a peticiones y respuestas, como los servidores de web habilitados para Java. Un servlet toma los datos de una forma HTML y aplicarles las reglas del negocio para actualizar la base de datos de la compañía. Los servlets son para los servidores lo que los applets son para los browsers o navegadores, pero los servlets carecen de *interfaz gráfica de usuario (GUI, Graphic User Interface)*. Son un efectivo sustituto de los CGI's.

A continuación, se explica más detalladamente cada una de las características del lenguaje de programación Java.

Simple:

Los creadores de Java se basaron en el lenguaje de programación C++, pero eliminaron muchas de las características orientadas a objetos que se utilizaban esporádicamente o casi nunca. C++ es un lenguaje para programación orientada a objetos y ofrece características muy importantes. Sin embargo, como sucede con muchos lenguajes diseñados para parecer poderosos, algunas características a menudo provocan problemas a los programadores, como la creación de códigos que frecuentemente contienen errores y que apenas se logran entender. Puesto que la mayor parte del costo de la Ingeniería de Software se encuentra en el mantenimiento del código más que en su creación, el cambio a un código comprensible en lugar de uno con más opciones pero difícil de comprender puede ayudar a reducir el costo del software.

Específicamente, Java se diferencia de C++ (y de C) en los siguientes aspectos:

1. Java no ofrece soporte a los tipos de datos `struct`, `union` y `pointer`.
2. Java no ofrece soporte a `typedef` ni a `#define`.
3. Java se diferencia en su manejo de ciertos operadores y no permite la sobrecarga de operadores.
4. Java no ofrece soporte a herencia múltiple.
5. Java maneja los argumentos de la línea de comandos de manera diferente a C o C++.
6. Java tiene una clase `String` como parte del paquete `java.lang`. Esto difiere de los arreglos de caracteres de terminación nula, como se utiliza en C y en C++.
7. Java tiene un sistema automático para asignación y liberación de memoria (recolector de basura), de manera que es innecesario utilizar las funciones de asignación y desasignación, como en C y en C++.

Se puede aprender a programar en Java *rápidamente* una vez que se comprendan los conceptos básicos de la programación orientada a objetos. Java realiza un esfuerzo para no tener características sorprendentes. Hay muchos sistemas de programación que se enorgullecen de su versatilidad la hora de proporcionar docenas de maneras diferentes de realizar lo mismo. Si un lenguaje muestra lo suficiente las interioridades de la máquina, será libre de hacer casi cualquier cosa, de la manera que desee. Aunque es cierto que esto ofrece un rendimiento impresionante para el programador muy cuidadoso y experto, la libertad tiene un precio en cuanto a complejidad y compresión. En Java hay un número reducido de maneras de realizar una tarea dada.

Este estilo de simplicidad ha producido a menudo "lenguajes tipo guión" ineficientes y no expresivos. Java no es un lenguaje tipo guión. Estos lenguajes le quitan la posibilidad de innovar, asumiendo que cualquier comportamiento que pueda desear ya está encapsulado en los objetos incorporados que simplemente hay que enumerar. Java le da la posibilidad de expresar toda idea que tenga de una manera orientada a objetos directa y limpia, sin tener que exponer todas las interioridades del sistema subyacente.

Orientado a objetos: Como C++, Java puede dar soporte a un enfoque orientado a objetos para la escritura de software. De manera ideal, el diseño orientado a objetos permite la creación de componentes de software que pueden volverse a utilizar.

Técnicamente, las características orientadas a objetos de Java son las de C++ con extensiones de Objetive C para resolución dinámica de métodos.

Es sorprendente ver la cantidad de dialectos nuevos de lenguajes antiguos que se describen a sí mismos como orientados a objetos. Parece que se lanza muy a menudo esta palabra típica, al igual que "mejorado para Internet". Java fue

diseñado partiendo de cero. No es un derivado directo de otro lenguaje de programación y no es compatible con ninguno de ellos.

Debido a que existió la libertad de diseñar a partir de un pizarrón en blanco, se eligió un enfoque para los objetos limpio, utilizable y pragmàtico. Java ha tomado prestadas con libertad ideas de muchos entornos de software de objetos pioneros de las últimas décadas, con lo que consigue un equilibrio entre el modelo purista "todas las cosas son objetos" y el modelo "quédate fuera de mi camino" de los dedicados a saltarse las barreras de seguridad (hackers). El modelo de objeto en Java es simple y fácil de ampliar, mientras que los números y otros tipos simples se mantienen como "no objetos" de alto rendimiento.

Las mayorías de los otros sistemas orientados a objetos han elegido tener jerarquías de objetos rígidas y difíciles de gestionar, o utilizan modelos de objeto completamente dinámicos que renuncian al rendimiento y facilidad de compresión. Java consigue de nuevo un equilibrio, proporcionando un mecanismo de clasificación simple con un modelo de interfaz dinámica intuitiva sólo donde resulta necesario. La compresión del estilo de programación orientado a objetos es el obstáculo más alto que hay que superar para aprender a programar en Java y no es realmente muy alto.

Distribuido:

A diferencia de los lenguajes C++ y C, Java se diseñó específicamente para trabajar en un ambiente de redes. Java contiene una biblioteca muy grande de clases para la comunicación utilizando el grupo de protocolos de TCP/IP, incluyendo protocolos como HTTP y FTP. El código de Java puede manipular recursos a través de URL, con la misma facilidad con que los programadores acostumbran tener acceso a un sistema local de archivos utilizando C++ y C.

Interpretado:

Java realiza esta extraordinaria proeza de independencia de plataforma compilando en una representación intermedia llamada "*código de byte Java*" (*bytecodes*, en inglés), que se puede interpretar en cualquier sistema que tenga un intérprete de Java.

Cuando el compilador de Java traduce un archivo fuente de clase a código de byte (*bytecodes*), este archivo de clase puede ejecutarse en cualquier máquina que dé soporte a un visualizador que funciona con Java o que tenga un sistema operativo que soporte Java. Esto permite que el código de Java se escriba independientemente de la plataforma y elimina para el cliente el ciclo de compilación y ejecución, porque los códigos de byte (*bytecodes*) no son específicos para una máquina determinada sino que se interpretan.

Otros sistemas independientes de plataforma también son sistemas interpretados. Estos lenguajes sufren déficit de rendimiento casi insalvables. Sin embargo Java fue diseñado para ejecutarse bien en CPU de muy poca potencia. Aunque es cierto que Java es interpretado, el código de byte de Java fue diseñado con cuidado de manera que fuese sencillo traducirlo directamente a código máquina nativo para conseguir un muy alto rendimiento. Los sistemas intérpretes de Java que realizan esta optimización "*justo a tiempo*" (*Just In Time, JIT* por sus siglas en inglés) no pierden ninguno de los beneficios del código neutral a la plataforma.

Sólido:

Un software sólido no se "quiebra" fácilmente por errores de programación. Un lenguaje de programación que estimula un software sólido coloca más restricciones para el programador cuando éste escribe código fuente. Tales restricciones incluyen los tipos de datos y el uso de apuntadores. El lenguaje de programación C se relaja notablemente, es decir, es muy poco estricto en la verificación de tipos de datos. En Java, por ejemplo, la tipificación es más rigurosa: un programador no puede efectuar una conversión forzada (*cast*) de un entero arbitrario en un apuntador. Además, Java no

ofrece soporte a apuntadores aritméticos, en cambio tiene arreglos. Esta simplificación elimina algunos de los trucos que los programadores de C podían utilizar para tener acceso a áreas arbitrarias de la memoria. En particular, Java no permite que el programador sobreescriba en la memoria ni corrompa otros datos a partir de apuntadores. Como contraste, un programador en C a menudo puede escribir o corromper datos por accidente (o de manera deliberada).

Robusto: Java le restringe en unas cuantas áreas clave para obligarle a encontrar pronto sus errores en el desarrollo del programa. A la vez, Java le libera de tener que preocuparse por muchas de las causas principales de error del programador en la mayoría de los otros lenguajes. Java verifica su código a la vez que lo escribe, y una vez más antes de ejecutarlo para asegurarse. Muchos de los errores difíciles de encontrar que se convierten a menudo en situaciones en tiempo de ejecución difíciles de reproducir son imposibles de crear en Java. El poder estar seguro de lo que se ha escrito se comportará de una manera predecible bajo condiciones diversas es una de las características claves de Java.

Muchos de los programas que se utilizan hoy en día fallan por una de las siguientes razones: gestión de memoria o condiciones excepcionales. Son dos elementos muy importantes, dado que es imposible escribir un programa útil sin tener que gestionar memoria o generar condiciones excepcionales de alguna manera. La gestión de memoria es una tarea desagradable en entornos tradicionales que obliga al programador a controlar toda la memoria asignada y asegurarse de liberarla al sistema cuando ya no la utiliza. A menudo los programadores se olvidan de liberar la memoria que han asignado, o todavía peor, intentan liberar memoria que otra parte de su código espera continuar utilizando. Las condiciones excepcionales en los entornos tradicionales surgen a menudo en situaciones como una división por cero o un "archivo no encontrado", y deben ser gestionadas mediante construcciones torpes y difíciles de leer. Java elimina

virtualmente ambos problemas con una gestión de memoria avanzada llamada "recolector de basura", y un manejo de excepciones orientado a objetos integrados. Esta libertad de no tener que realizar cantidades masivas de contabilidad le permite al usuario ser productivo sin ningún temor.

Debido a que Java es un lenguaje muy estricto en cuanto a tipos y declaraciones, la mayoría de los errores típicos se pueden descubrir durante la compilación. Esto supone un ahorro de tiempo comparado con tener que ejecutar el programa y comprobar todas sus partes antes de encontrar un error de ejecución dinámico. Es un ejemplo donde Java es menos flexible que algunos otros lenguajes.

Seguro: Como Java funciona en ambientes de redes, el tema de la seguridad debe interesar de manera particular a los programadores. Actualmente es muy popular la seguridad, especialmente cuando se refiere a Internet.

Uno de los principios de diseño claves de Java es la seguridad. Java nunca ha tenido características inseguras que ahora se tengan que remediar para hacerlo seguro. La mayoría de las maneras obvias de saltarse la barrera de seguridad de un sistema no se pueden describir como un programa de Java. Dado que los programadores de Java no pueden llamar a funciones globales y ganar el acceso a recursos arbitrarios del sistema, se puede ejercer un control sobre los programas ejecutables de Java que no es posible en otros sistemas. Los límites en apuntadores no le permiten a los programadores falsificar el acceso a la memoria cuando éste no se permite. Tales características de Java hacen posible un ambiente de software más seguro.

De arquitectura neutral: Los diseñadores de Java tomaron varias decisiones difíciles en el lenguaje Java y el intérprete, de modo que pueda realmente "escribir una vez, ejecutar en cualquier sitio, en cualquier momento y para siempre". Simplemente tiene que preocuparse por escribir un buen programa y Java se asegurará que

funcione en Macintosh, PC, Unix y lo que puedan ofrecer las plataformas futuras.

El compilador de Java crea código de byte que se interpreta en esa máquina, que cuenta con el intérprete de Java o que tiene instalado un visualizador que funciona con Java.

Portable: Al ser una arquitectura neutral se permite una gran capacidad de portabilidad. Sin embargo otro aspecto de la portabilidad es la manera en que el hardware interpreta las operaciones aritméticas. En C y C++ el código fuente puede ejecutarse de manera ligeramente diferente en diferentes plataformas de hardware, debido a la manera en que estas plataformas implantan las operaciones aritméticas. En Java, esto se ha simplificado. Un tipo entero en Java, `int`, es un complemento a dos, con signo, de un entero de 32 bits. Un número real, `float`, siempre es un número de punto flotante de 32 bits, definido por el estándar IEEE 754.

De alto desempeño: Como los códigos de byte (*bytecodes*) son interpretados, el proceso de programación a veces no es tan rápido como la compilación y la ejecución directas en una plataforma de hardware en particular. La compilación de Java incluye una opción para traducir los códigos de bytes (*bytecodes*) en código de máquina para la plataforma específica de hardware. Esto puede ofrecer la misma eficiencia que un proceso de compilación y cargado tradicional. De acuerdo con las pruebas de Sun Microsystems, el desempeño de esta traducción de código de byte a código de máquina es "casi indistinguible" de la compilación directa de programas en C o C++.

Multihilos: Java es un lenguaje que se puede utilizar para crear aplicaciones en las que suceda más de una cosa a la vez. Basado en un sistema de rutinas que permiten múltiples "hilos" de eventos a partir del paradigma de condición y el monitor de C. A. R. Hoare, Java le presenta al programador una manera de apoyar la conducta en tiempo real e interactiva en programas.

Dinámico: A diferencia del código C++, que a menudo exige que se vuelva a compilar por completo si cambia una clase madre, Java utiliza un método de interfaces para aligerar esta dependencia. El resultado es que los programas de Java pueden permitir nuevos métodos y variables de instancia en una biblioteca de objetos, sin afectar los objetos cliente dependientes.

Interactivo: Java fue diseñado para cumplir el requisito del mundo real de crear programas en red interactivos. La mayoría de los sistemas pasan un mal trago tratando a la vez con interactividad y la transmisión por red. Java tiene características avanzadas que le permiten escribir programas que hacen muchas cosas al instante, sin perder el rastro de lo que debería suceder y cuándo. El problema con hacer más de una cosa a la vez es mantener sincronizadas todas las partes que dependen entre sí. El intérprete de Java viene con la solución más elegante diseñada hasta ahora para sincronización entre múltiples procesos, y hace que sea posible construir sistemas interactivos que se ejecutan sin problemas. Los hilos múltiples de Java de utilización sencilla le permiten pensar en el comportamiento específico que se intenta codificar, sin tener que integrar ese comportamiento en un modelo de programación global de "bucle de suceso".

Como se comentó en páginas anteriores, Java es toda una plataforma de cómputo integrada, principalmente, por el lenguaje de programación Java, la *Máquina Virtual de Java* (*Java Virtual Machine, JVM*) y las librerías y *API's* (*Application Programming Interfaces*). A continuación se analizará la máquina virtual de Java y el modelo de seguridad que maneja ese lenguaje.

4.1.3 La Máquina Virtual de Java (JVM, Java Virtual Machine).

Java está diseñado para maximizar la portabilidad. Muchos detalles sobre Java están definidos específicamente para todas las implementaciones. Muchos lenguajes dejan las definiciones exactas a las implementaciones concretas y ofrecen sólo garantías generales,

como un rango mínimo, o proporcionan una forma de preguntar al sistema qué rango hay en la plataforma actual.

Java desciende en la especificación de estas definiciones hasta el lenguaje máquina al que se traduce el código Java. El código fuente Java se compila en *códigos byte*, diseñados para ejecutarse en una *máquina virtual Java*. Los códigos byte son un lenguaje máquina para una máquina abstracta, pero son interpretados por la máquina virtual en cada uno de los sistemas que soporta Java.

La habilidad de Java para bajar, integrar y ejecutar código de una computadora remota es una espada de doble filo. Por el lado positivo, el uso de Java permite a una computadora obtener nuevas capacidades con poca intervención del usuario. En suma, Java no requiere instalación a nivel de disco duro ni conexión de archivos dudosamente seguros. En el lado negativo, de cualquier modo, la maquinación compleja de Java deja una computadora vulnerable al ataque. Un applet java hostil, que sea cargado a través de la red en el disco duro del host cliente, podría realizar actividades maliciosas como ejecutar comandos locales y eliminar archivos del disco.

Los diseñadores de Java hicieron lo mejor para que tales actividades maliciosas sean imposibles, implementando un modelo de seguridad. Este modelo de seguridad ejecuta un número de verificaciones antes de permitir a un applet ejecutarse.

La seguridad de Java se fundamenta en tres niveles de defensa: *El Verificador de código de byte o bytecode*, *El Cargador de clases applet* y *el Manejador de seguridad*. Juntos, estos tres niveles de seguridad, ejecutan el cargado y la verificación en tiempo real para restringir el acceso al sistema de archivos y a la red, así como restringir el acceso a visualizadores internos. Cada uno de los niveles depende, en alguna manera, de los otros dos. Cada parte deberá hacer su trabajo adecuadamente para que el modelo de seguridad funcione correctamente, como puede observarse en la figura siguiente (Figura 4-3).

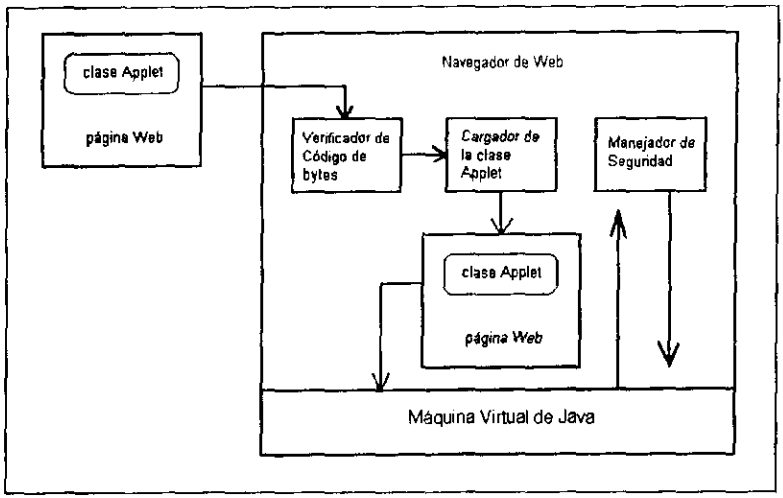


Figura 4-3

La figura anterior muestra los tres niveles de defensa dentro del marco de trabajo de Java. *El Verificador de código de bytes* es el primer nivel en el modelo de seguridad de Java. Cuando un programa fuente Java es compilado, es convertido a código de bytes Java independiente de la plataforma. El verificador, entonces, revisa que el código "indigno de confianza" se apegue a las reglas antes de que se le permita ejecutarse.

El verificador revisa el código de byte a un número diferente de niveles. La prueba más simple asegura que un archivo `.class` (por ejemplo, un archivo de código de bytes) tenga el formato correcto. En un nivel menos básico, el verificador aplica un teorema probador preconstruido a cada método. El teorema probador ayuda a asegurar que el código de byte no falsifique apuntadores, viole restricciones de acceso a objetos utilizando un tipo de información incorrecto. El proceso de verificación, en conjunto con la definición del lenguaje Java, ayuda a establecer un conjunto base de garantías de seguridad.

El segundo nivel de seguridad de Java es *el Cargador de clases applet*. Típicamente proporcionado por un revisador de páginas, que carga todos los applets y las clases a las que éstos se refieren. Cuando un applet es cargado de la red, el Cargador de clases applet recibe el dato binario y se refiere a él como una nueva clase. El Cargador de clases determina cuándo y cómo un applet puede agregar clases a un ambiente java en ejecución. Parte del trabajo del Cargador de clases es asegurarse que el applet no instalará código que reemplace componentes importantes del ambiente java en tiempo de ejecución.

En general, un ambiente de ejecución java puede tener muchos cargadores de clase activos, cada uno definiendo su propio espacio de nombre. Los espacios de nombres permiten que las clases java estén separadas en distintos tipos, de acuerdo a dónde ellos se originaron. En otras palabras, un espacio de nombre es una porción de tipo seguro de la memoria con clases que están asociadas con un cargador de clases específico.

El tercer nivel del modelo de seguridad java es el Manejador de seguridad, el cual restringe la manera en la que el applet puede utilizar interfaces visibles. Por lo tanto, el manejador de seguridad implementa una buena porción del modelo de seguridad completo. Es un módulo sencillo que verifica. en tiempo de ejecución, métodos peligrosos tales como de acceso a la red o de acceso a archivos o aquellos en los que se definan nuevos Cargadores de clases.

El código de la biblioteca java consulta al Manejador de seguridad cuando una operación peligrosa vaya a ser probada. El manejador de seguridad tiene, entonces, una oportunidad para prohibir la operación generando una excepción de seguridad. Las decisiones realizadas por el manejador de seguridad toman en cuenta que el cargador de clases, precisamente, cargue las clases pedidas. Las clases preinstaladas localmente tienen más privilegios que las clases que se cargan de la red (como, por ejemplo, los applets); con ello se asegura no saturar la red para obtener un archivo que ya se encuentre localmente en la computadora cliente.

Además, la naturaleza del lenguaje Java indica que los programas denominados applets que se distribuyen a través de la red mundial Internet, tienen prohibido conectarse a otra computadora diferente de la cual se originan y de la cual se distribuyen.

A pesar de que se ha dicho que la máquina virtual Java es un conjunto de software, es decir, una máquina abstracta esto no impide que un sistema pueda implementar la máquina virtual Java en silicio utilizando un chip para fines específicos. Esto es lo que hoy en día se conoce como Java Chips y no afecta a la portabilidad de los códigos byte, es solamente una puesta en práctica de máquina virtual más.

La máquina virtual Java, las librería y las API's vienen integradas en el Kit de Desarrollo Java (Java Development Kit, JDK) como se muestra en la figura 4-4.

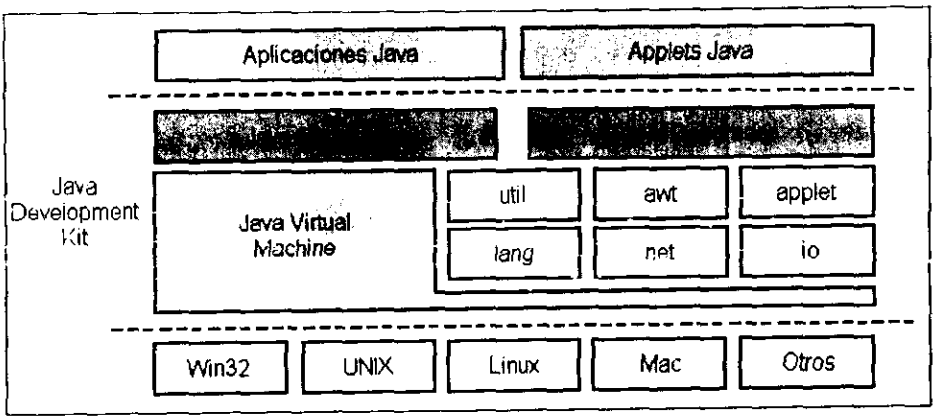


Figura 4-4

Como se observa, estamos hablando de tres niveles:

1. Las aplicaciones de usuario, que pueden ser applets, aplicaciones, etc.
2. El kit de desarrollo Java, que consta de un compilador, un depurador, la librería de clases y la máquina virtual Java.
3. La plataforma, que es transparente tanto para el programador como para el usuario final.

Por todo lo anterior, se decidió adoptar a Java como el lenguaje de programación con el que se desarrollará parte importante de nuestro sistema.

4.2 Los Sistemas Manejadores de Bases de Datos (DBMS)

Anteriormente a las bases de datos, en la industria de la computación se hacía uso de los archivos de datos pero éstos eran difíciles de manejar debido a las siguientes desventajas:

- *Redundancia de la información*, es decir, se tenía que volver a introducir o solicitar información que ya se había solicitado previamente.
- *Inconsistencia*, es decir, que existen archivos que manejan la misma información pero en algunos de ellos dicha información está actualizada y en otros no.
- *No eran concurrentes*, es decir, que no era posible que varios usuarios tuvieran acceso a la misma información al mismo tiempo.

Entonces surge el concepto *base de datos*. Una base de datos es un conjunto de información organizada cuya finalidad es servir a una o más aplicaciones de la mejor manera posible. Históricamente, las corporaciones utilizan las bases de datos para resolver problemas de negocios. Las bases de datos almacenan y manejan datos de manera que permiten seguridad, acceso y control de la información.

Las consideraciones que se deben hacer para la selección de la base de datos son el modelo y la topología de las bases de datos. El modelo incluye la estructura del almacenamiento de los datos y describe cómo serán presentados los datos al usuario o programador así como las relaciones entre los elementos de una base de datos, mientras que la topología incluye la configuración física tanto de datos como del procesamiento.

Las bases de datos se clasifican, con respecto a su modelo, en:

- Flat-file
- Jerárquico
- De Red
- Relacional

Mientras que, de acuerdo a su topología, las bases de datos se clasifican en:

- Centralizadas
- De *Red de Area Local* (*Local Area Network*, *LAN*) de computadoras personales.
- Sistemas Cliente/Servidor

Un sistema de bases de datos es un conjunto de *datos* que son los valores registrados físicamente en la base de datos, *hardware* que corresponde al almacenamiento físico, *software* que son los programas que manejan la base de datos y los *usuarios* que corresponden a los usuarios finales, el programador o desarrollador y el DBA que es la persona encargada de la creación y administración de la base de datos.

Desventaja de un sistema de bases de datos:

- La seguridad y la integridad pueden ser contraproducentes sin buenos controles.

Ventajas de un sistema de bases de datos:

- Reducir la redundancia.
- Evitar la inconsistencia.
- Aplicar restricciones de seguridad.
- Control de la concurrencia.

Un *Sistema Manejador de Bases de Datos* (*Database Management System*, *DBMS*) está formado por:

- La base de datos
- El software para manipular (almacenar, recuperar y modificar) los datos.

El principal objetivo de un DBMS es crear un ambiente en el que sea posible guardar y recuperar información de la base de datos en forma eficiente.

Respecto al Sistema Manejador de Bases de Datos, existen muchos de éstos en la industria de las bases de datos que se pueden utilizar para implementar soluciones de desarrollo. Cada uno de estos productos proporciona varias características que lo hacen único. Los principales factores en la elección de alguno de ellos son las características y el precio de cada uno.

En este desarrollo no se realizó una elección de entre varios DBMS ya que en el laboratorio de la División de Ingeniería Eléctrica (DIE) ya se contaba con anterioridad con el DBMS de Sybase llamado SQL Server versión System 10, el cual es un manejador relacional de bases de datos.

Este DBMS está instalado en la DIE desde el año de 1996 y es aquí donde se encuentran las tablas que contienen la información de cada usuario registrado en el laboratorio y que son los usuarios que van a hacer uso del sistema a desarrollar. Dicha información consta del nombre del usuario, dirección, carrera, login de acceso a la red DIE y su respectiva contraseña o password de validación. Se tiene contemplado que el sistema a desarrollar tenga acceso al login y password de cada usuario. Basándose en ello, se podrá validar o rechazar la entrada de ese usuario al sistema.

4.2.1 Principales DBMS existentes en la industria de las bases de datos

A pesar de lo anterior, aquí se explican brevemente otros DBMS y, posteriormente, se explican las características del DBMS de Sybase Inc.

DB2

DB2 es el sistema de bases de datos relacional de IBM y es uno de los DBMS relacionales más antiguos del mercado. Se utiliza principalmente en sistemas de computadoras mainframe como AS/400 y RS/6000. Este DBMS proporciona muchas características avanzadas y se utiliza, principalmente, para soluciones de bases de datos a gran escala.

Informix

El DBMS relacional Informix, de Informix Software Inc., está disponible tanto para las plataformas Unix como para Windows NT. Este DBMS se utiliza más para aplicaciones en un rango que va desde pequeñas a medianas, pero se puede emplear para proyectos de desarrollo mayores. Recientemente, Informix Software presentó su nuevo producto Object Relacional, el cual proporciona un manejo de objetos en la base de datos. Este nuevo tipo de bases de datos, orientadas a objetos, mejorará en gran medida las soluciones que puedan ofrecer los desarrolladores a sus clientes.

Oracle

Para muchos, el líder en el mercado de las bases de datos es el DBMS Oracle de Oracle Corporation. Es el DBMS relacional de uso más extendido y ofrece varias características que facilitan su uso. Recientemente Oracle liberó su versión 8.0, la cual es un DBMS relacional orientado a objetos.

Microsoft SQL Server

La versión de SQL Server de Microsoft ha sido mejorada para su uso en aplicaciones reales de bases de datos y se incluye en el paquete BackOffice. Este DBMS es, en cierta forma, menos costoso que el resto de los productos de bases de datos y se utiliza más para el desarrollo de aplicaciones pequeñas.

4.2.2 El DBMS de Sybase Inc.

Inicialmente el DBMS de Sybase se llamaba SQL Server y era soportado por múltiples plataformas como UNIX, Windows y OS/2. Posteriormente, Sybase cambia el nombre el nombre de su manejador por el de Adaptive Server Enterprise (ASE) agregándole así nuevas características que lo colocan como uno de los DBMS más utilizados en la industria.

Actualmente se distribuye principalmente en dos versiones: el Adaptive Server Anywhere (ASA) y Adaptive Server Enterprise (ASE). Este último es el adecuado para ambientes corporativos ya que presenta una gran robustez, mientras que el ASA es más adecuado a

computadoras personales y portátiles en las que se desea un DBMS sólido y robusto como el de Sybase.

Entre las principales características que presenta, están las siguientes:

- Ambiente SQL estándar.
- Arquitectura multiprocesos y abierta.
- El SQL Server Monitor que ayuda a visualizar y optimizar el performance del sistema.
- Detección y resolución automática de bloqueos del sistema (deadlocks).
- Procedimientos almacenados locales y remotos Servidor-a-Servidor.
- Soporte al cómputo distribuido a través del Sybase SQL Server Manager.
- Escalabilidad a través de aplicaciones y datos que se pueden portar fácilmente vía el SQL Anywhere.
- Mantiene la integridad y seguridad de los datos a través de respaldos y duplicidad de bases de datos así como de la codificación de passwords.
- Fácil instalación y mantenimiento.

5. La arquitectura Cliente-Servidor distribuida

La solución es desarrollar un sistema que cubra las necesidades especificadas en un principio y que a la vez, sea eficiente y trabaje en ambientes de red.

Por lo tanto, estamos hablando de un manejador de bases de datos al que se pueda tener *acceso desde una terminal* o computadora, que puede ser remota, para que en ella pueda ejecutarse la aplicación.

Se decidió investigar lo referente a la arquitectura cliente-servidor, predominante en estos días y que cubre las expectativas de la mayoría de los sistemas distribuidos así como de los ambientes de red.

5.1 Comunicaciones en ambientes de red e Introducción a la arquitectura Cliente-Servidor

El modelo Cliente-Servidor es el modelo estándar de ejecución de aplicaciones en una red. Un modelo cliente/servidor para sistemas de computadoras conectadas en red se relaciona con tres componentes: el cliente, el servidor y la red. Un servidor es un proceso que se está ejecutando en un nodo de la red (terminal o periférico conectado a la red) y que gestiona el acceso a un determinado recurso. Un cliente es un proceso que se ejecuta en el mismo o en diferente nodo y que realiza peticiones de servicio al servidor. Las peticiones están originadas por la necesidad de tener acceso al recurso que gestiona el servidor (Figura 5-1).

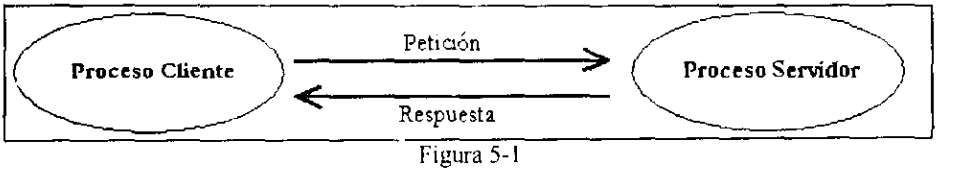


Figura 5-1

El sistema de Entrada/Salida de Unix sigue el paradigma que normalmente se designa como Abrir-Leer-Escribir-Cerrar. Antes de que un proceso de usuario pueda realizar operaciones de entrada/salida, debe hacer una llamada a Abrir (open), para indicar y obtener permisos para su uso, del archivo o dispositivo que quiere utilizar. Una vez que el objeto está abierto, el proceso de usuario realiza una o varias llamadas a Leer (read) y Escribir (write), para conseguir leer y escribir datos respectivamente. Leer toma datos desde el objeto y los

transfiere al proceso de usuario, mientras que Escribir transfiere datos desde el proceso de usuario al objeto. Una vez que todos estos intercambios de información estén concluidos, el proceso de usuario llamará a Cerrar (close) para informar al sistema operativo que ha finalizado la utilización del objeto que antes había abierto.

Cuando se incorporan las características a Unix de comunicación entre procesos (IPC) y el manejo de redes, la idea fue implementar la interface con IPC similar a la que se estaba utilizando para la entrada/salida de archivos, es decir, siguiendo el paradigma del párrafo anterior. En Unix, un proceso tiene un conjunto de descriptores de entrada/salida desde donde Leer y por donde Escribir. Estos descriptores pueden estar referidos a archivos, dispositivos, o canales de comunicaciones (sockets). El ciclo de vida de un descriptor, aplicado a un canal de comunicación (socket), está determinado por tres fases (siguiendo el paradigma):

- Creación y apertura del socket.
- Lectura y escritura, recepción y envío de datos por el socket.
- Destrucción, cierre del socket.

La interface IPC en Unix-BSD está implementada sobre los protocolos de red TCP y UDP. Los destinatarios de los mensajes se especifican como direcciones de socket; cada dirección de socket es un identificador de comunicación que consiste en una dirección Internet y un número de puerto.

Las operaciones IPC se basan en pares de sockets. Se intercambian información transmitiendo datos a través de mensajes que circulan entre un socket en un proceso y otro socket en otro proceso. Cuando los mensajes son enviados, se encolan en el socket hasta que el protocolo de red los haya transmitido. Cuando llegan, los mensajes son encolados en el socket de recepcion hasta que el proceso que tiene que recibirlos haga las llamadas necesarias para recoger esos datos.

5.1.1 Sockets Stream, Datagrama y Raw

Los sockets son puntos finales de enlaces de comunicaciones entre procesos. Los procesos los tratan como descriptores de archivos, de forma que se pueden intercambiar datos con otros procesos que se están transmitiendo y recibiendo a través de sockets. El tipo de sockets describe la forma en la que se transfiere información a través de ese socket.

Sockets Stream (TCP, Transmission Control Protocol)

Son un servicio orientado a conexión donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados.

El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

Sockets Datagrama (UDP, User Datagram Protocol)
Son un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la confiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

Sockets Raw
Son sockets que dan acceso directo a la capa de software de red subyacente o a protocolos de más bajo nivel. Se utilizan sobre todo para la depuración del código de los protocolos.

5.1.2 Diferencias entre sockets Stream y Datagrama

Ahora se presenta una decisión: ¿Qué protocolo, o tipo de sockets, debemos usar — UDP o TCP? La decisión depende de la aplicación cliente/servidor que estemos escribiendo. Vamos a ver algunas diferencias entre los protocolos para ayudar en la decisión.

En UDP, cada vez que se envía un datagrama, hay que enviar también el descriptor del socket local y la dirección del socket que va a recibir el datagrama, luego éstos son más grandes que los TCP. Como el protocolo TCP está orientado a conexión, tenemos que establecer esta conexión entre los dos sockets antes de nada, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no existe en UDP.

En UDP hay un límite de tamaño de los datagramas, establecido en 64 kilobytes, que se pueden enviar a una localización determinada, mientras que TCP no tiene límite; una vez

que se ha establecido la conexión, el par de sockets funciona como los streams: todos los datos se leen inmediatamente, en el mismo orden en que se van recibiendo.

UDP es un protocolo desordenado, no garantiza que los datagramas que se hayan enviado sean recibidos en el mismo orden por el socket de recepción. Al contrario, TCP es un protocolo ordenado, garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado.

Los datagramas son bloques de información del tipo "*lanzar y olvidar*". Para la mayoría de los programas que utilicen la red, el usar un flujo TCP en vez de un datagrama UDP es más sencillo y hay menos posibilidades de tener problemas. Sin embargo, cuando se requiere un rendimiento óptimo, y está justificado el tiempo adicional que supone realizar la verificación de los datos, los datagramas son un mecanismo realmente útil.

En resumen, TCP parece más indicado para la puesta en práctica de servicios de red como un control remoto (rlogin, telnet) y transmisión de archivos (ftp); que necesitan transmitir datos de longitud indefinida. UDP es menos complejo y tiene una menor sobrecarga sobre la conexión; esto hace que sea el indicado en la puesta en práctica de aplicaciones cliente/servidor en sistemas distribuidos montados sobre redes de área local.

5.1.3 Uso de Sockets, Puertos y Servicios

Podemos pensar que un servidor en Internet es un conjunto de sockets que proporciona capacidades adicionales del sistema, los llamados servicios. Cada servicio está asociado a un puerto. Un puerto es una dirección numérica a través de la cual se procesa el servicio. Sobre un sistema Unix, los servicios que proporciona ese sistema se indican en el archivo `/etc/services` y algunos ejemplos son:

Servicio	**Puerto/Protocolo**	**Alias**
Daytime	13/udp	
ftp	21/tcp	
telnet	23/tcp	telnet
smtp	25/tcp	mail
http	80/tcp	

La primera columna indica el nombre del servicio. La segunda columna indica el puerto y el protocolo que está asociado al servicio. La tercera columna es un alias del servicio; por

ejemplo, el servicio smtp, también conocido como *mail*, es la puesta en práctica del servicio de correo electrónico.

Las comunicaciones de información relacionadas con el Web tienen lugar a través del puerto 80 mediante protocolo TCP. Para emular esto en Java, usaremos la clase *Socket* propia de Java, pero sería diferente con la fecha (daytime) ya que el servicio que toma la fecha y la hora del sistema, está ligado al puerto 13 utilizando el protocolo UDP. Un servidor que lo emule en Java usaría un objeto DatagramSocket.

5.1.4 Dominios de comunicaciones: Dominio Unix y Dominio Internet

El mecanismo de sockets está diseñado para ser todo lo genérico posible. El socket por sí mismo no contiene información suficiente para describir la comunicación entre procesos. Los sockets operan dentro de dominios de comunicación, dentro de estos dominios se define si los dos procesos que se comunican se encuentran en el mismo sistema o en sistemas diferentes y cómo pueden ser direccionados.

Dominio Unix

Bajo Unix, hay dos dominios, uno para comunicaciones internas al sistema y otro para comunicaciones entre sistemas.

Las comunicaciones intrasistema (comunicaciones entre dos procesos en el mismo sistema) ocurren en el dominio Unix. Se permiten tanto los sockets stream como los datagrama. Los sockets de dominio Unix bajo Solaris 2.x se implementan sobre TLI (Transport Level Interface). En el dominio Unix no se permiten sockets de tipo Raw.

Dominio Internet

Las comunicaciones intersistemas proporcionan acceso a TCP, ejecutándose sobre el protocolo IP (Internet Protocol). De la misma forma que el dominio Unix, el dominio Internet permite tanto Sockets Stream como Datagrama, pero además permite sockets de tipo Raw.

Los sockets Stream permiten a los procesos comunicarse a través de TCP. Una vez establecidas las conexiones, los datos se pueden leer y escribir hacia/desde los sockets como un flujo (stream) de bytes. Algunas aplicaciones de servicios TCP son:

- File Tranfer Protocol, FTP
- Simple Mail Transfer Protocol, SMTP

- TELNET, servicio de conexión de terminal remota

Los sockets datagrama permiten a los procesos utilizar el protocolo UDP para comunicarse hacia y desde esos sockets por medio de bloques. UDP es un protocolo no fiable y la entrega de los paquetes no está garantizada. Servicios UDP son:

- Simple Network Management Protocol, SNMP
- Trivial File Transfer Protocol, TFTP (versión de FTP sin conexión)
- Versatile Message Transaction Protocol, VMTP (servicio fiable de entrega punto a punto de datagramas independiente de TCP)

Los sockets raw proporcionan acceso al Internet Control Message Protocol, ICMP y se utiliza para comunicarse entre varias entidades IP.

5.1.5 La arquitectura Cliente-Servidor distribuida

Además de comentar la arquitectura cliente servidor tradicional, podemos hablar de la *arquitectura cliente servidor distribuida* o *arquitectura de ambientes distribuidos*. Esta arquitectura consiste en separar la arquitectura de 2 niveles en *una arquitectura de n niveles o capas*. Comúnmente, se habla de arquitecturas de 3 capas en la que la primer capa o nivel es el cliente, la segunda se refiere al servidor de aplicaciones o *middleware* que recibe las peticiones del cliente y la tercer capa corresponde al servidor de bases de datos que recibe la petición del middleware, que para ello se convirtió en cliente del servidor de bases de datos. Entonces el servidor de bases de datos envia como resultado la información necesaria al middleware y éste a su vez la envía como respuesta al cliente (Figura 5-2).

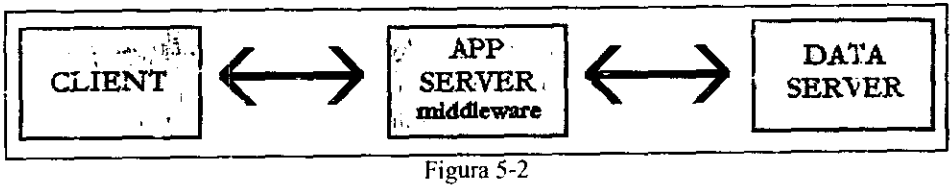


Figura 5-2

Esta arquitectura está desplazando a la tradicional cliente servidor debido a las siguientes razones:

- Encapsula objetos y aprovecha la herencia.
- Reutilización de objetos.

- Facilidad de mantenimiento.
- Hace un uso más efectivo de la red, ya que sólo se transportan datos validados y se realizan peticiones válidas al servidor de bases de datos.
- Aumenta la productividad al apoyarse en: metodologías orientadas a objetos, herramientas CASE, lenguajes conocidos, el encapsulamiento de procesos y la estandarización de interfaces.
- Integración a sistemas existentes.
- Es una arquitectura multiplataforma.

Ahora se analizará cada uno de los tres niveles de este sistema: el cliente, el servidor de aplicaciones y el servidor de bases de datos.

5.2 El Cliente y el Servidor de aplicaciones

Debido a que se implementará la arquitectura de ambientes distribuidos en este sistema, se requiere considerar a los procesos cliente y servidor de aplicaciones. El proceso cliente de este sistema es el correspondiente a la interfaz gráfica de usuario (GUI) en la que el usuario, inicialmente, introduce su login y password. Entonces el cliente se comunica a través de sockets con el servidor de aplicaciones para enviarle el login y el password del usuario. Una vez que el servidor ha recibido estos datos, se comunica con el servidor de bases de datos para enviarle el login y el password del usuario para que sea verificado en la tabla en la cual se encuentra esa información propia de cada usuario.

Una vez que el servidor ha verificado estos datos, envía al servidor de aplicaciones la respuesta, ya sea de aceptación o rechazo, y una vez que el servidor recibe dicha respuesta, la envía al cliente. Cuando se acepta la entrada del usuario, entonces se despliega el mapa del laboratorio con todas las computadoras. Para cada acción que el usuario desee realizar, el cliente se debe comunicar con el servidor de aplicaciones y éste con el servidor de bases de datos ya que la información referente al estado de todas las computadoras y a los apartados de los usuarios se encuentra en tablas de la base de datos.

Por lo tanto, se está hablando de tener una aplicación Java en el servidor de aplicaciones y que sea la que se conecte con el cliente y con el servidor de bases de datos. El lenguaje de programación Java, por naturaleza, se integra al trabajo en red de sockets y al estándar de ambientes distribuidos en varias capas. Ahora, se analizará el modelo de comunicaciones en Java a través de sockets.

5.2.1 Modelo de comunicaciones con Java

En Java, crear una conexión socket TCP/IP se realiza directamente con el paquete `java.net`. La integración de Java con la red se ha hecho legendaria, gracias en gran parte a la bibioteca de clase `java.net`. Java admite el protocolo TCP/IP de Internet ampliando la interfaz de E/S de flujo propia del lenguaje en el paquete `java.io` y añadiendo las características que se necesitan para crear objetos de E/S a través de la red. Java admite las familias de protocolo TCP (Stream) y UDP (Datagrama). A continuación se muestra un diagrama de lo que ocurre en el lado del cliente y del servidor Java cuando hay una comunicación a través de sockets. (Figura 5-3).

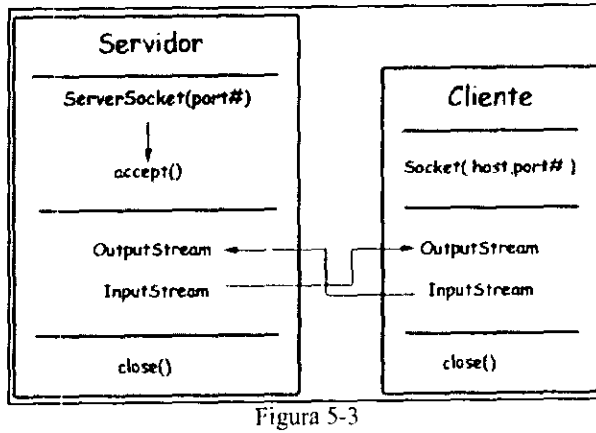


Figura 5-3

El modelo de sockets más simple es:

- El servidor establece un puerto `port#` y espera a que el cliente establezca la conexión. Cuando el cliente solicite una conexión, el servidor abrirá la conexión socket con el método `accept()`.
- El cliente establece una conexión con el host pasado como parámetro en `host` a través del puerto que se designe en `port#`.
- El cliente y el servidor se comunican con manejadores `InputStream` y `OutputStream` que son clases de Java que están contenidas en el paquete `java.io`.

Hay una cuestión al respecto de los sockets, que viene impuesta por la puesta en práctica del sistema de seguridad de Java. Actualmente, los applets sólo pueden establecer conexiones con el nodo desde el cual se transfirió su código. Esto está implementado en el JDK y en el intérprete de Java de los navegadores de web. Esto reduce en gran manera la

flexibilidad de las fuentes de datos disponibles para los applets pero elimina el problema de que si se permite que un applet se conecte a cualquier máquina de la red, entonces se podrían utilizar los applets para inundar la red desde una terminal con un cliente web del que no se sospecha y sin ninguna posibilidad de rastreo.

5.2.2 JDBC: El estándar del lenguaje Java para la conexión a las bases de datos

Debido a que el servidor de aplicaciones se conectará a un servidor de bases de datos, es necesario que la aplicación Java que residirá en el servidor de aplicaciones pueda conectarse a una base de datos y entienda su lenguaje, el SQL (Strutured Query Language).

Una de las desventajas principales de la versión 1.0.2 de Java era que no tenía soporte alguno para el acceso a bases de datos. Esto limitó la utilidad de Java en el campo de los negocios. Sin embargo, a partir de la versión 1.1 del JDK, Java proporciona un soporte completo para bases de datos por medio de JDBC (Java Database Connectivity). JavaSoft, la subsidiaria de Sun Microsystems referente a Java, introdujo JDBC el cual permite que los programas Java se conecten a cualquier base de datos utilizando diversos controladores (conocidos también como drivers) y un nuevo conjunto de objetos y métodos de la API (Interfaz de Programación de Aplicaciones) de Java (Figura 5-4).

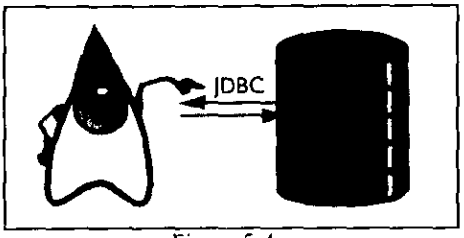


Figura 5-4

JDBC provee diversos objetos para conectarse a una base de datos, manipular datos y ejecutar instrucciones de SQL (Structured Query Language) y procedimientos almacenados; también proporciona varios objetos y métodos para obtener información de las tablas de la base de datos.

Hasta antes de la API JDBC, la única forma de acceso a bases de datos desde el Web fue a través de los programas CGI (Common Gateway Interface). JDBC es mucho más rápido que los scripts CGI. El uso de CGI por lo regular requiere que usted llame a otro programa que tiene que ser ejecutado por la computadora. Este programa intermedio realiza el acceso

a la base de datos, procesa los datos y los envía en un flujo al programa que realiza la llamada. Esto requiere de varios niveles de procesamiento lo que a su vez incrementa el tiempo de espera así como también propicia que aparezcan más errores.

Invocar un script CGI implica en realidad ejecutar un nuevo script, por lo regular a través de un servidor Web, mientras que la ejecución de una instrucción SQL a través de JDBC en la base de datos requiere sólo de cierto tipo de servidor que pase los comandos SQL a través de la base de datos. Esto abrevia en forma muy notable el tiempo que se necesita para ejecutar las instrucciones de SQL. Mientras que el script CGI debe, de hecho, conectarse a la base de datos y procesar los resultados, la solución JDBC permite que su programa Java tenga la conexión a la base de datos para, de esa manera, manejar todo el procesamiento.

La API JDBC define clases Java para representar conexiones a bases de datos, sentencias SQL, conjuntos de resultados, metadatos de bases de datos, etc. Además permite al programador escribir en el lenguaje de programación Java y que resultará en sentencias SQL y procesar los resultados. JDBC es la API primaria para acceso a bases de datos en el lenguaje de programación Java.

Los drivers JDBC caen en una de las siguientes cuatro categorías:

Driver tipo 1: ***El bridge JDBC-ODBC (JDBC-ODBC bridge)*** provee accesos JDBC vía la mayor parte de los drivers ODBC. Es importante notar que el código binario de algunos drivers ODBC, y en muchos casos el código del cliente de la base de datos también, deberá ser cargado en cada máquina cliente que utiliza este driver, por lo que este tipo de driver es más apropiado en una red corporativa (Figura 5-5a).

Driver tipo 2: ***Un driver basado en tecnología Java parcialmente con API nativo (native-API partly_Java technology-based driver)*** convierte las llamadas JDBC en llamadas de la API del cliente para Oracle, Sybase, Informix, DB2 u otros DBMS (Sistemas Manejadores de Bases de Datos). Es importante notar que este tipo de driver requiere que algún código binario sea cargado en cada máquina cliente (Figura 5-5b).

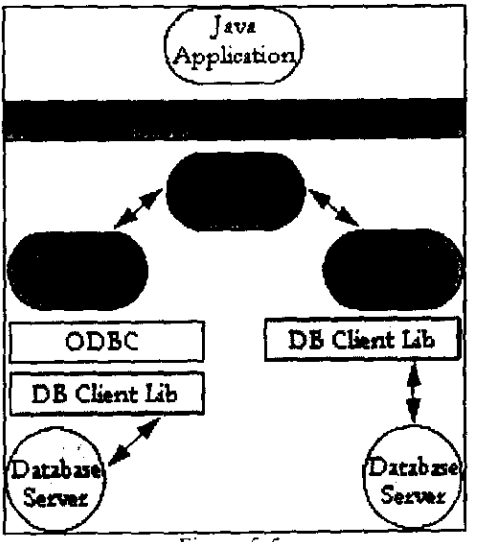


Figura 5-5

Driver tipo 3: ***Un driver basado en tecnología Java totalmente con protocolo de red (net-protocol all-Java technology-based driver)*** traslada las llamadas de JDBC en un protocolo de red independiente del DBMS y dicho protocolo es, entonces, trasladado a un protocolo DBMS por un servidor. Este servidor de red intermedio (middleware) está habilitado para conectar sus clientes basados en tecnología Java con muchas diferentes bases de datos. El protocolo de red específico depende del proveedor. En general, esta es la alternativa JDBC más flexible ya que los proveedores de los drivers tienen productos que soportan el acceso a Internet e Intranets así como requerimientos adicionales de seguridad, acceso a través de muros de fuego (firewalls), etc. Además, varios proveedores están agregando drivers JDBC a sus productos middleware de bases de datos existentes (Figura 5-6a).

Driver tipo 4: ***Un driver basado en tecnología Java totalmente con protocolo nativo (native-protocol all-Java technology-based driver)*** convierte las llamadas JDBC a un protocolo de red utilizado por el DBMS directamente. Esto permite una llamada directa de la máquina cliente al servidor DBMS y es una solución práctica para el acceso a Intranet. Debido a que los vendedores de bases de datos son proveedores de estos protocolos, ellos mismos son la principal fuente de este estilo de driver por lo que todavía está en desarrollo (Figura 5-6b).

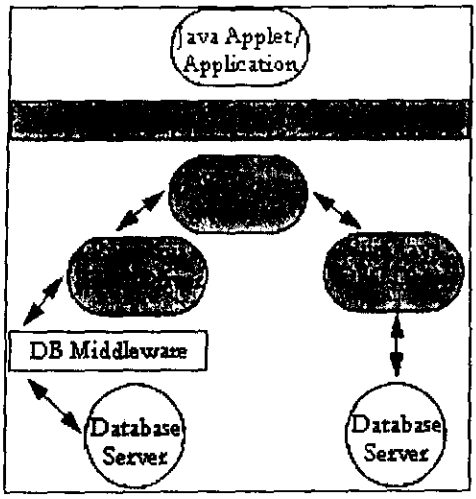


Figura 5-6

El servidor de bases de datos contiene un DBMS (Data Base Management System, Sistema Manejador de Bases de Datos) tal como Oracle, Sybase, Microsoft SQL Server, Informix, DB2, etc. Para este sistema, se utilizó el driver tipo 4 de Sybase Inc., llamado **JConnect**. Las características que presenta este driver son las siguientes:

- Soporte al estándar JDBC.
- Driver 100 % Java Puro
- Soporte a protocolos nativos.
- No requiere instalación del cliente del DBMS.
- El tamaño del archivo de código de bytes es pequeño para una rápida transferencia.
- Configuración para ambientes de red multiniveles.
- Soporte a SSL y a murallas de fuego (firewalls).
- Alto performance para acceso directo a bases de datos.
- Soporte a diversas bases de datos
- Cero administración
- Interoperable con otros drivers JDBC.
- Soportado por cualquier cliente habilitado para Java y servidores de web.
- Escalable para aplicaciones críticas de negocio.
- Seguro

5.3 Integración del sistema con sistemas existentes

En un principio estaba contemplado crear tanto la interfaz gráfica para los usuarios del laboratorio como la interfaz gráfica de los recepcionistas del laboratorio. Ambas interfaces debían ser distintas ya que la interfaz para los recepcionistas del laboratorio debía constar con opciones de entrada y salida de los usuarios, renovación del uso de una computadora, llegada de un usuario que previamente hubiera apartado una computadora, entre otras cosas y que no deben venir en la interfaz de los usuarios del laboratorio ya que son acciones exclusivas de los recepcionistas del laboratorio.

Durante el análisis y diseño de este sistema, el creador del sistema actual de apartados del laboratorio decidió realizar la interfaz gráfica de su sistema por lo que el nuevo sistema no desplazará al sistema actual sino que se integrará con él y, por lo tanto, se tuvieron nuevas consideraciones para la creación de dicho sistema:

- Además de manejar la información a través de tablas de una base de datos, había que realizar el manejo de la información en archivos, ya que el sistema actual así maneja la información y para que ésta sea consistente, el nuevo sistema compartirá los archivos con ese sistema actual para, posteriormente, realizar una especie de "respaldo" de la información en las tablas de la base de datos.

- Se omite la creación de una interfaz para los recepcionistas del laboratorio.

- La computadora que se encuentra en la recepción y que contiene el sistema actual, deberá contener al nuevo sistema y soportar un servidor de web para que los usuarios remotos (clientes web) puedan establecer una conexión al sistema a desarrollar.

Por lo tanto, el esquema que muestra la arquitectura de la integración de los dos sistemas quedará como se muestra en la siguiente figura (Figura 5-7).

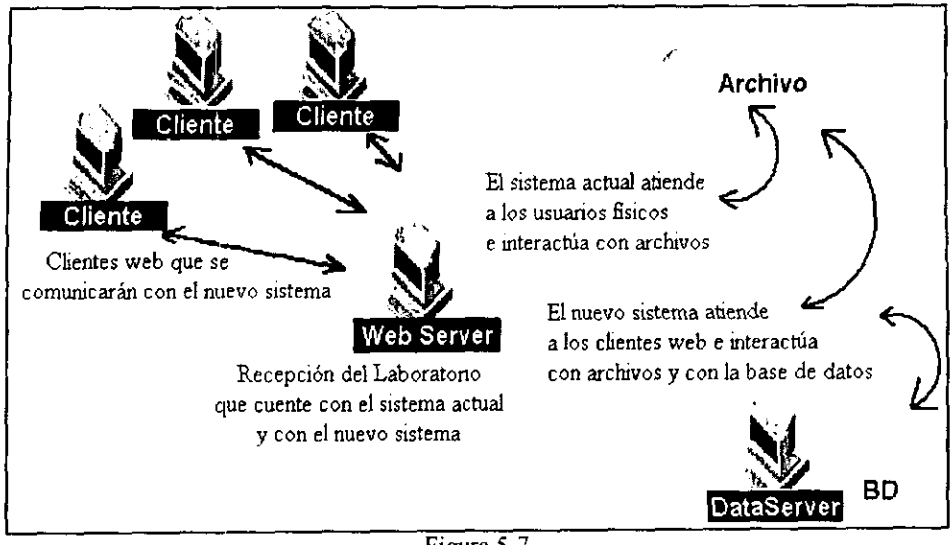


Figura 5-7

Ahora se realizará la programación correspondiente a cada uno de los niveles que conforman este sistema.

6. Programación del sistema

6.1 Programación de sockets y flujos de E/S

Apertura de Sockets

Las clases referentes a sockets se encuentran en el paquete `java.net`. Si estamos programando un cliente, el socket se abre de la forma:

```
Socket miCliente;
miCliente = new Socket( "maquina",numeroPuerto );
```

Donde `maquina` es el nombre de la máquina en donde estamos intentando abrir la conexión y `numeroPuerto` es el puerto (un número entero) del servidor al cual nos queremos conectar. Cuando se selecciona un número de puerto, se debe tener en cuenta que los puertos en el rango 0-1023 están reservados para usuarios con muchos privilegios (superusuarios o root). Estos puertos son los que utilizan los servicios estándar del sistema como `email`, `ftp` o `http`. Para las aplicaciones que se desarrollen, asegurarse de seleccionar un puerto superior al 1023.

En el ejemplo anterior no se utilizan excepciones; sin embargo, es una gran idea la captura de excepciones cuando se está trabajando con sockets. El mismo ejemplo quedaría como:

```
Socket miCliente;
try {
    miCliente = new Socket( "maquina",numeroPuerto );
} catch( IOException e ) {
    e.printStackTrace();
  }
```

Si estamos programando un servidor, la forma de apertura del socket es creando un objeto `ServerSocket`, como se muestra en el siguiente ejemplo:

```
ServerSocket miServicio;
try {
    miServicio = new ServerSocket( numeroPuerto );
} catch( IOException e ) {
    e.printStackTrace();
  }
```

A la hora de la puesta en práctica de un servidor también necesitamos crear un objeto socket de la clase `Socket` para que esté atento a las conexiones que le puedan realizar clientes potenciales y poder aceptar esas conexiones:

```
Socket socketServicio = null;
try {
    socketServicio = miServicio.accept();
} catch( IOException e ) {
    e.printStackTrace();
  }
```

Creacion de Streams

- **Creación de Streams de Entrada**

En la parte cliente de la aplicación, se puede utilizar la clase `InputStream` para crear un flujo de bytes de entrada para recibir todas las respuestas que el servidor le envíe. Además podemos utilizar la clase `InputStreamReader` que sirve de puente de flujos de bytes a flujos de caracteres. Una vez obtenido el flujo de caracteres, se puede utilizar la clase `BufferedReader` que en su constructor recibe un objeto de la clase `InputStreamReader`.

```
BufferedReader entrada;
try {
    entrada = new BufferedReader( new InputStreamReader(
                                        miCliente.getInputStream() ) );
} catch( IOException e ) {
    e.printStackTrace();
  }
```

La clase `BufferedReader` permite la lectura de líneas de texto y tipos de datos primitivos de Java de un modo altamente portable; dispone del método `readLine()` que regresa un objeto `String`.

En el lado del servidor, también se utilizará la misma estructura, pero en este caso para recibir las entradas que se produzcan de los clientes que se hayan conectado:

```
BufferedReader entrada;
try {
    entrada = new BufferedReader( new InputStreamReader(
                                    socketServicio.getInputStream() ) );
} catch( IOException e ) {
    e.printStackTrace();
  }
```

- **Creación de Streams de Salida**

En el lado del cliente, podemos crear un flujo de salida para enviar información al socket del servidor utilizando las clases `PrintWriter` y `OutputStream`. `PrintWriter` representa el flujo de salida de objetos en modo texto y utiliza para su constructor un objeto `OutputStream` existente, en este caso se obtiene del objeto `Socket`.

```
PrintWriter salida;
try {
    salida = new PrintWriter( miCliente.getOutputStream(),true );
} catch( IOException e ) {
    e.printStackTrace();
  }
```

En el código anterior se crea un nuevo `PrintWriter` para un existente `OutputStream` y a través de un objeto `OutputStreamWriter` convierte los caracteres en bytes utilizando la codificación predeterminada de caracteres. Este flujo es habilitado solamente cuando se invoca al método `println()`.

Para el lado del servidor, se realiza la misma estructura para enviar información a los clientes:

```
PrintWriter salida;
try {
   salida = new PrintWriter( socketServicio.getOutputStream(),true );
} catch( IOException e ) {
    e.printStackTrace();
  }
```

Cierre de Sockets

Siempre deberemos cerrar los canales de entrada y salida que se hayan abierto durante la ejecución de la aplicación. En la parte del cliente se tienen los siguientes métodos:

```
try {
    salida.close();
    entrada.close();
    miCliente.close();
} catch( IOException e ) {
    e.printStackTrace();
    }
```

Y en la parte del servidor se tiene una estructura similar:

```
try {
    salida.close();
    entrada.close();
    socketServicio.close();
    miServicio.close();
} catch( IOException e ) {
    e.printStackTrace();
    }
```

Clases útiles en comunicaciones

Se van a exponer otras clases que resultan útiles cuando estamos desarrollando programas de comunicaciones, aparte de las que ya se han visto. Se agrega también el nombre completo de la clase, es decir, se indica a qué paquete pertenece.

- `java.io.IOException`
  Señala que una excepción de Entrada/Salida, de algún ordenamiento, ha ocurrido.

- `java.net.InetAddress`
  Esta clase representa una dirección IP (Internet Protocol). Las aplicaciones deberán utilizar los métodos `getLocalHost()`, `getByName()` o `getAllByName()` para crear una nueva instancia de `InetAddress`.

- `java.net.URL`

  La clase URL representa un Localizador Uniforma de Recursos (Uniform Resource Locator) que es un apuntador a un recurso del World Wide Web. Un recurso puede ser algo tan simple como un archivo o un directorio o un objeto más complicado como una consulta a una base de datos. Por ejemplo:

  ```
  http://www.ncsa.uiuc.edu/demoweb/url-primer.html
  ```

  En general, un URL puede ser descompuesto en varias partes. El ejemplo previo indica que el protocolo utilizado es `http` (HyperText Transport Protocol) y que la información reside en una máquina host llamada `www.ncsa.uiuc.edu`. La información en esa máquina se nombra `demoweb/url-primer.html`. El significado exacto de este nombre es dependiente tanto del host como del protocolo.

  Un URL puede, opcionalmente, especificar un puerto el cual es el número del puerto por el cual las conexiones TCP se realizarán en ese host remoto. Si el puerto no se

especifica, se utiliza el puerto predeterminado. Por ejemplo, el puerto predeterminado para `http` es el `80`. Los puertos alternativos pueden ser especificados de la siguiente manera:

```
http://www.ncsa.uiuc.edu:8080/demoweb/url-primer.html
```

- `java.net.UnknownHostException`

  Esta excepción es generada para indicar que la dirección IP de un host no puede ser determinado.

- `java.net.MalformedURLException`

  Esta excepción se genera para indicar que un URL mal formado ha ocurrido debido a que el protocolo no es válido o que la cadena que forma este URL no puede ser analizada.

6.2 Programación del servidor de aplicaciones

La programación del servidor de aplicaciones se va a realizar en el lenguaje de programación Java utilizando JDK en su versión 1.1.7. Este lenguaje da soporte a la programación orientada a objetos, facilita la conexión con sockets y bases de datos y una aplicación escrita en Java es capaz de ejecutarse sobre cualquier plataforma. A continuación se presenta un fragmento de código:

```
/**
  * Server.java para JDK1.1
  *
  */

... // Se importan los paquetes necesarios

// Inicia la clase Server, la cual es hija de la clase Thread
// y su funcion es escuchar por el puerto 4444 del servidor y cuando
// recibe la peticion de un cliente, crea otro hilo que espera por
// una peticion de otro cliente
public class Server extends Thread
{
 // Empieza la declaracion de los campos cuyo ambito
 // es toda la clase en si.
 // serverSocketControl es el objeto de la clase ServerSocket que
 // "abre" el puerto para recibir el mensaje de control, es decir, que
 // servicio del servidor desea el cliente
 private static ServerSocket serverSocketControl=null;
 // serverSocketDatos es el objeto de la clase ServerSocket que
 // "abre" el puerto para recibir los datos
```

```
 private static ServerSocket serverSocketDatos=null;
 // clientSocketControl es el objeto de la clase Socket que "recibe"
 // una peticion
 private Socket clientSocketControl=null;
 // clientSocketDatos es el objeto de la clase Socket que "recibe"
 // datos del cliente
 private Socket clientSocketDatos=null;
 // compu es un campo de tipo int que indica si una computadora ha sido
 // ocupada o apartada. 0 si la computadora esta libre y 1 en caso
contrario.
 private static int compu=0;
 // apartadas es un array de char que contiene los ceros o unos de las
 // computadoras, dependiendo de si estan libres u ocupadas.
 private static char[] apartadas;
 // num_compus es un campo de tipo int que obtiene el numero de comps.
 // existentes en el lab.
 private static int num_compus;
 // usuario es un array de 2 Strings que contiene el login y el password
 // de cada usuario que va a ingresar al sistema
 private static String usuario[]=new String[2];
 // fechaActual es un objeto de la clase GregorianCalendar y que contiene
 // la hora exacta actual con el formato
 // año, mes, dia, hora, minutos, segundos
 private GregorianCalendar fechaActual=null;
 private int yearFechaActual=0;
 private int monthFechaActual=0;
 private int dayFechaActual=0;
 private int hourFechaActual=0;
 private int minutesFechaActual=0;
 private int secondsFechaActual=0;
 private GregorianCalendar horaInicio=null;
 private GregorianCalendar horaTermino=null;

 ...

 public static void main(String[] args)
 {
  try
  {
   num_compus=Integer.parseInt(args[0])+1;
   apartadas=new char[num_compus];
   for(int i=0;i<apartadas.length;i++)
    apartadas[i]='0';
   serverSocketControl=new ServerSocket(4443);
   System.out.println("Puerto 4443 para control");
   serverSocketDatos=new ServerSocket(4444);
   System.out.println("Puerto 4444 para datos");
  }
  catch(ArrayIndexOutOfBoundsException e)
  {
   System.out.println("La sintaxis es: java Server
<numero_de_computadoras>");
   System.exit(1);
  }
  catch (IOException e)
  {
   System.err.println("Error en los puertos");
   System.exit(1);
  }
  Server objeto1=new Server();
  objeto1.start();
```

```
 }
 ...
}
```

Como se observa en el código anterior, se utilizan dos puertos de comunicaciones, el 4443 para control de peticiones y el 4444 para envío y recepción de datos.

Además el número total de computadoras en el laboratorio se coloca como argumento necesario al momento de iniciar la aplicación. Con esto, el número de computadoras que puede manejar este sistema es variable y puede cambiar sin necesidad de modificar los archivos fuente ni recompilar.

Se utilizó la clase `java.util.GregorianCalendar` para el manejo de fechas y horas.

Para la conexión a la base de datos, se requiere tener el driver previamente instalado y unas líneas de código similar a las del siguiente fragmento:

```
// url es el objeto String que tiene como valor la direccion y el puerto
// del servidor de bases de datos asi como la base de datos en la cual
// se encuentran las tablas necesarias para la consulta.
// La sintaxis de la linea siguiente es especifica de cada driver
private static String url="jdbc:sybase:Tds:132.248.59.2:1468/proyectos";

...

public String confirmaEntrada(String[] arreglo)
{
 String mensaje="Login incorrecto";
 try
 {
  String query="select password from usuario where
login='"+arreglo[0]+"'";
  Class.forName("com.sybase.jdbc.SybDriver");
  Connection conexion=DriverManager.getConnection(url,login,password);
  Statement sentencia=conexion.createStatement();
  ResultSet resultado=sentencia.executeQuery(query);
  while(resultado.next())
  {
   if(arreglo[1].equals(resultado.getString(1)))
   {
    mensaje="Login valido";
    ...
   }
   ...
  }
  ...
 }
 ...
}
```

En la línea `Class.forName("com.sybase.jdbc.SybDriver")` se "carga" la clase cuyo nombre que se pasa como argumento y que es la clase del driver *JConnect* de *Sybase*. Esta clase debe estar declarada dentro de la variable de ambiente `CLASSPATH` para que sea reconocida por el JDK. Las clases `Connection`, `Statement` y `ResultSet` se encuentran en el paquete `java.sql`.

6.3 Programación del cliente

La programación del cliente se va a realizar en el lenguaje Java utilizando el JDK en su versión 1.1.7. Este lenguaje posee toda una jerarquía de clases capaz de implementar cualquier elemento gráfico que se requiera la cual se encuentra organizada dentro del paquete `java.awt` del lenguaje Java.

Durante el análisis y diseño del cliente, se manejaron algunas clases que van a conformar la interfaz gráfica que van a utilizar los usuarios. Esas clases son las siguientes:

- PasswordDialog
- Ventana de mensaje
- Mapa de computadoras

Java cuenta con la clase `java.awt.Dialog` por lo que la clase `PasswordDialog` va a ser una clase hija de ésta. Las características que debe reunir esta ventana de validación son las siguientes:

- Debe ser una ventana que no sea redimensionable, es decir, que el usuario no pueda modificar su tamaño.
- Debe ser una ventana tipo modal, es decir, que el usuario no puede ejecutar nada en el navegador de web hasta que dicha ventana no sea cerrada y para ello, tiene dos opciones: una es haber sido validado por el sistema y la otra opción es cancelar su entrada al sistema.
- Va a constar de 2 etiquetas de texto, una para indicar el login y otra para indicar el password del usuario, así como 2 campos de texto, uno para que el usuario escriba su login y el otro para su password y dos botones, uno para aceptar la información proporcionada por el usuario y otro para cancelar su entrada, como se mostró en la figura 3-10 del capítulo 3.

En Java, la herencia se especifica con la palabra reservada `extends`. Es importante señalar que es necesario importar las clases de los paquetes java que vienen con el JDK con la palabra reservada `import`, como se muestra a continuación.

```
import java.awt.Button;
import java.awt.Dialog;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Label;
import java.awt.Rectangle;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

// Inicia la clase PasswordDialog que crea una ventana no redimensionable
// y que cuenta con objetos AWT para la validacion del usuario
// al sistema

class PasswordDialog extends Dialog implements
                        ActionListener,KeyListener,FocusListener
{
 private Label loginLabel;
 private Label passwordLabel;
 private TextField loginTextField;
 private TextField passwordTextField;
 private Button okButton,cancelButton;
```

En el fragmento anterior se importan las clases necesarias que debe utilizar esta clase y se inicia la declaración de la clase con la palabra reservada `class` seguida del nombre de la clase y posteriormente se especifica que esta clase se hereda de la clase `Dialog` que se encuentra en el paquete `java.awt`. En ese mismo paquete se encuentran las clases `Button`, `Label` y `TextField` con las cuales se crearon los seis objetos que va a contener esta ventana. Debido a que dichos objetos deben responder a eventos de teclado o mouse, es necesario importar algunas clases e interfaces del paquete `java.awt.event` como las clases `ActionEvent` (para eventos del mouse), `FocusEvent` (para eventos del foco o cursor) y `KeyEvent` (para eventos del teclado) y sus respectivas interfaces que reciben y manejan los eventos generados. Las interfaces se implementan en esta clase con la palabra reservada `implements` seguida de los nombres de las interfaces separadas por coma (,).

Para inicializar las variables de esta clase (los seis objetos anteriores) se creó un constructor de la clase `PasswordDialog`. Un constructor es un método que se llama igual que la

clase y que se utiliza para la inicialización de objetos ya que se va a invocar cada vez que se cree una instancia (objeto) de esa clase. El constructor va a tener como parámetros un objeto `java.awt.Frame`, una cadena de caracteres para el título de la ventana y una cadena de caracteres de la dirección IP del servidor de aplicaciones al cual se va a conectar para enviarle el login y el password del usuario. Dicho servidor recibirá estos datos y se conectará a la base de datos para validar o rechazar al usuario.

```
public PasswordDialog(Frame parent,String title,String host)
{
```

El objeto parent de la clase `Frame` se va a pasar como parámetro al invocar el constructor de la clase padre (la clase `java.awt.Dialog`) así como el título de la ventana y un valor booleano que indica si dicha ventana será modal (true) o no (false). El constructor de la clase padre se invoca desde el constructor de la clase hija a través de la palabra reservada super.

```
  super(parent,title,true);
  setResizable(false);
  this.host=host;
```

Se desea que al presionar el botón con la etiqueta "Aceptar" se realice la conexión al servidor de aplicaciones y se envíen a éste el login y el password del usuario que desea entrar al sistema. Para ello, debe agregarse la siguiente línea a la declaración del objeto `Button`:

```
okButton.addActionListener(this);
```

El método `addActionListener()` agrega el módulo especificado de escucha que recibirá los eventos de acción para este botón. Dichos eventos ocurren cuando el botón es presionado o liberado con el mouse. Además debe implementarse el método `actionPerformed()`, el cual va a contener el siguiente código:

```
public void actionPerformed(ActionEvent event)
 {
  Button botonEvento=(Button)event.getSource();
  if(botonEvento.getLabel().equals("Aceptar"))
  {
   if(loginTextField.getText().trim().length()==0 ||
passwordTextField.getText().trim().length()==0)
    new AlertDialog(new Frame(),"Mensaje de sistema","Login y password
incorrectos");
   else
   {
    String usuario[]=new String[2];
    usuario[0]=loginTextField.getText();
```

```
    usuario[1]=passwordTextField.getText();

   // Si no se pasa una direccion de un servidor, entonces toma
   // un predeterminado
   if(this.host==null)
    this.host="132.248.59.98";

   // AQUI SE CONECTA AL SERVIDOR DE APLICACIONES Y LE PIDE QUE
   // SE CONECTE LA BASE DE DATOS PARA VERIFICAR AL USUARIO
   try
   {
    socket=new Socket(InetAddress.getByName(this.host),4444);
    out=new PrintWriter(socket.getOutputStream(),true);
    out.println("entrada");
    for(int i=0;i<usuario.length;i++)
     out.println(usuario[i]);
    in=new BufferedReader(new
InputStreamReader(socket.getInputStream()));
    respuesta=in.readLine();
    if(respuesta.equals("Login valido"))
    {
     indicador="aceptado";
     this.setVisible(false);
     user=loginTextField.getText();
     cerrarConexion();
    }
    else
    {
     AlertDialog alerta=new AlertDialog(new Frame(),"Mensaje de
sistema",respuesta);
     cerrarConexion();
    }
   }
   catch(UnknownHostException e)
   {
    indicador="desconocido";
    String texto="Accion no disponible por el momento";
    AlertDialog alerta=new AlertDialog(new Frame(),"Mensaje de
sistema",texto);
    this.setVisible(false);
   }
   catch(IOException e)
   {
    indicador="desconocido";
    String texto="Accion no disponible por el momento";
    AlertDialog alerta=new AlertDialog(new Frame(),"Mensaje de
sistema",texto);
    this.setVisible(false);
   }
  }
  if(botonEvento.getLabel().equals("Cancelar"))
  {
   this.setVisible(false);
   System.out.println("Destruyendo el dialog");
   indicador="inaceptado";
  }
 }
```

La ventana de mensaje es la clase llamada `AlertDialog` que también es una clase hija de la clase `java.awt.Dialog`. En el fragmento de código anterior se observa que para crear una instancia de esta clase se invoca al constructor a través de la siguiente línea de código:

```
AlertDialog alerta=new AlertDialog(frame,"Mensaje de sistema",texto);
```

Y en la clase `AlertDialog`, se va a tener el siguiente código:

```
...
Label texto;
Button okButton;

// Constructor de la clase, que presenta una ventana con informacion
// y un boton de Aceptar
public AlertDialog(Frame parent,String titulo,String texto)
{
 super(parent,titulo,true);
 setBackground(Color.white);
 setResizable(false);
 ...
 // Etiqueta con el mensaje
 this.texto=new Label();
 this.texto.setText(texto);
 this.texto.setAlignment(Label.CENTER);
```

Con lo anterior, tenemos una clase reutilizable llamada `AlertDialog` que cuenta con un mensaje, en una línea de texto, que puede ser personalizado.

Una vez que se ha autorizado la entrada del usuario al sistema, debe aparecer una clase que va a contener el mapa de computadoras del laboratorio. Para ello, se creó la clase llamada `Mapa`, la cual es hija de la clase `java.applet.Applet` e implementa la interfaz `Runnable` para poder manejar varios procesos simultáneamente (es decir, va a ser una clase *multihilos* o *multithreading*).

Además, se creó otra clase sencilla que va a desplegar un reloj en cada cliente llamada `Reloj`. Con ello, se muestra la hora del servidor y se evitan los problemas de manejar el reloj de cada máquina cliente que podría no tener la hora ni la fecha correcta.

7. Resultados, instalación y pruebas

7.1 Resultados

El sistema completo consta de los siguientes archivos:

- El archivo `Server.java` genera el archivo `Server.class` que contiene a la clase `Server` que es el servidor de aplicaciones y se debe ejecutar en el host remoto servidor de Web.
- El archivo `num_conexion.dat` que es donde el servidor de aplicaciones guarda las conexiones realizadas por los usuarios, a manera de bitácora o archivo log.
- El archivo `actual.txt` que contiene las computadoras ocupadas en este momento.
- El archivo `apartadas.txt` que contiene las computadoras que están reservadas para ocuparse posteriormente.
- El archivo `Entrada.html` que se refiere a la clase `Entrada`.
- El archivo `Entrada.java` genera el archivo `Entrada.class` que contiene a la clase `Entrada`, cargada en el host cliente, la cual hace referencia a la clase `PasswordDialog` y dependiendo de ésta, muestra o no el archivo `Mapa.html`.
- El archivo `PasswordDialog.java` genera el archivo `PasswordDialog.class` que contiene a la clase `PasswordDialog` que es la ventana de validación de los usuarios al sistema. Dependiendo del resultado de la validación, se muestra o no la clase `AlertDialog`.
- El archivo `AlertDialog.java` genera el archivo `AlertDialog.class` que contiene a la clase `AlertDialog` la cual es la ventana de alerta con un mensaje del sistema.
- El archivo `Mapa.html` que se refiere a las clases `Mapa` y `Reloj`.
- El archivo `Mapa.java` genera el archivo `Mapa.class` que contiene a la clase `Mapa` la cual define el mapa de computadoras del laboratorio.
- El archivo `ImagePanel.java` genera el archivo `ImagePanel.class` que contiene a la clase `ImagePanel` la cual consiste en un panel para la imagen de una computadora.
- El archivo `Reloj.java` genera el archivo `Reloj.class` que contiene a la clase `Reloj` la cual consta de un reloj en tiempo real.
- El archivo `maquina.gif` que contiene la imagen de las computadoras en estado libre.
- El archivo `maq_ocup.gif` que contiene la imagen de las computadoras en estado apartado.

- El archivo `maq out.gif` que contiene la imagen de las computadoras fuera de servicio.
- El archivo `Logout.html` que se muestra cuando el usuario abandona el sistema.

7.2 Instalación

El sistema es realmente muy pequeño en cuanto a capacidad ya que los archivos necesarios ocupan una capacidad total de 101 KB aproximadamente

Todos los archivos deben colocarse en una computadora que cuente con las siguientes características:

Sistema operativo Windows 95, Windows 98, Windows NT, Linux, Unix, OS/2, etc Es decir, prácticamente cualquier sistema operativo que maneje ambientes de red
Un servidor Web instalado como por ejemplo, Apache, El servidor http de NCSA, Netscape Enterprise, Microsoft Personal Web Server, Microsoft Internet Information Server, etc.
Un intérprete de Java, para poder ejecutar la clase Server.

En cuanto a los requerimientos de hardware, éstos no difieren de los necesarios para contar con el sistema operativo que se haya elegido.

Por el lado del cliente, no se requiere un sistema operativo específico. Lo único necesario es que cuente con un navegador de Web compatible con Java en su versión 1.1. Tales navegadores pueden ser Netscape 4.5 o posterior, Internet Explorer 4.0 o posterior, Hot Java 1.1 o posterior, entre otros.

7.3 Pruebas

Las pruebas se realizaron en diversas computadoras, principalmente en una computadora que cuenta con procesador Pentium II, memoria RAM de 64 MB, sistema operativo Windows NT 4.0.

Para ejecutar el servidor se utiliza la siguiente línea de comando

```
java Server 44
```

y se despliegan las siguientes líneas:

```
Puerto 4443 para control
Puerto 4444 para datos
```

En este momento, el servidor está listo para recibir las peticiones de los clientes. En el lado del cliente se carga el archivo Entrada.html en el navegador de Web, como se muestra en la figura 7-1.

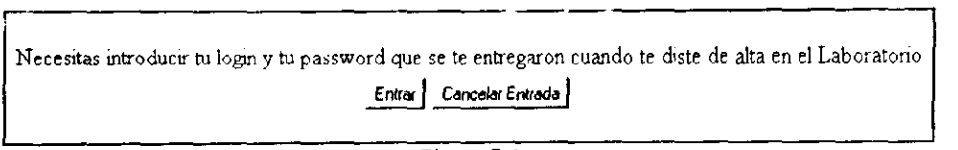


Figura 7-1

Al hacer clic sobre el botón con la etiqueta "Entrar", aparece la ventana de validación como se muestra en la figura 7-2.

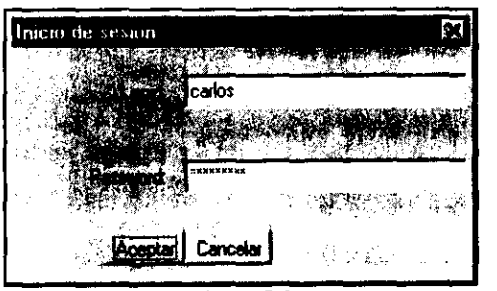


Figura 7-2

Si los datos son incorrectos, se muestra la ventana de alerta (Figura 7-3).

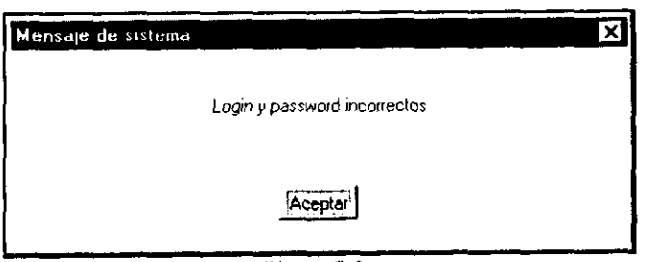


Figura 7-3

En caso de que sean correctos, el navegador de Web carga el archivo Mapa.html, como se muestra en la siguiente figura (Figura 7-4).

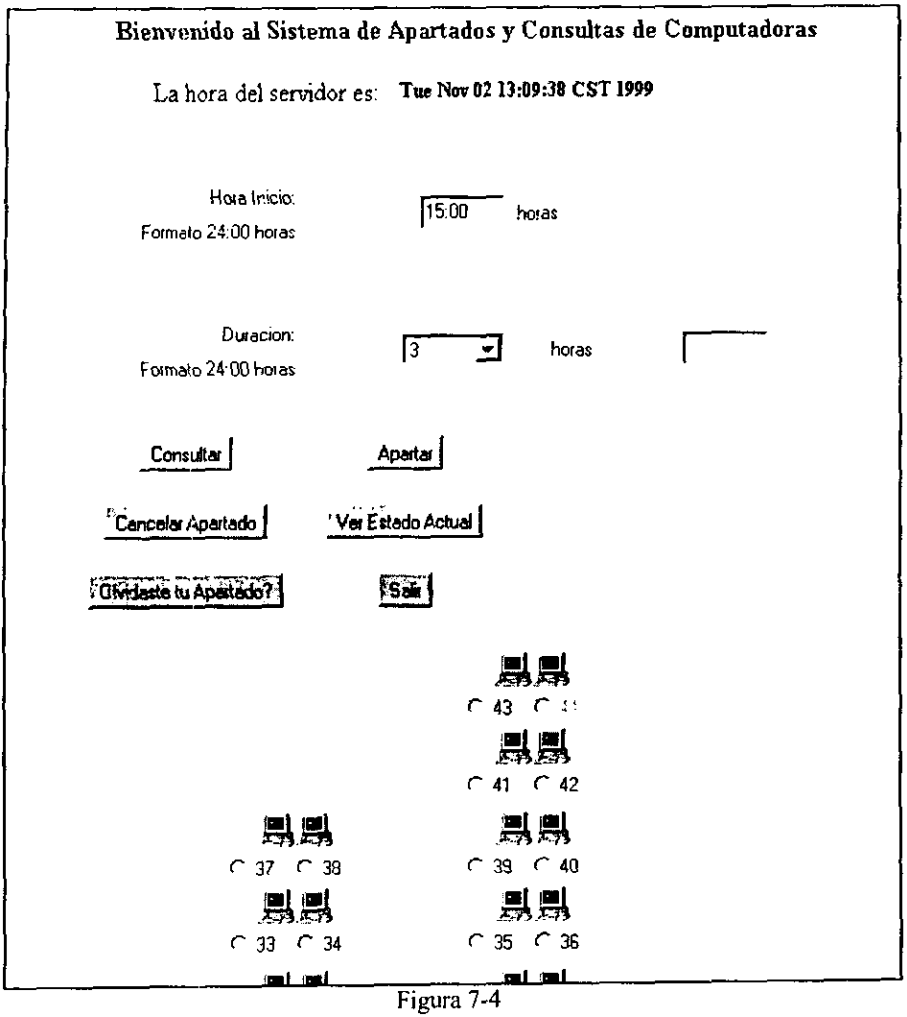


Figura 7-4

En este mapa de computadoras del laboratorio se muestran las computadoras libres en color azul, las computadoras ocupadas en color rojo y las computadoras fuera de servicio en color amarillo. Mientras tanto, el servidor acepta la conexión del cliente y empieza a enviarle información de las computadoras a éste:

```
Llego el cliente 200.33.175.138!
Enviando el cambio a los clientes: 0
La fecha actual es: Tue Nov 02 13:23:10 CST 1999
```

Si llegara a suceder un cambio de estado de una o más de las computadoras, éste es enviado a los clientes activos y las computadoras cambian de color. Además se envía a éstos la fecha y hora del servidor. La clase Reloj cargada en los navegadores de los clientes actualiza el reloj en tiempo real.

La interfaz de los clientes cuenta con un campo de texto para especificar la hora de inicio de la consulta o apartado y otro para la hora de término o en su defecto se puede especificar las horas cubiertas a través de un menú de selección. Posteriormente, se puede presionar el botón "*Consultar*" o seleccionar una computadora y presionar el botón "*Apartar*". En este último caso, si se cuenta con un apartado anterior, se muestra la ventana de alerta con un mensaje del sistema indicando una acción cancelada (Figura 7-5).

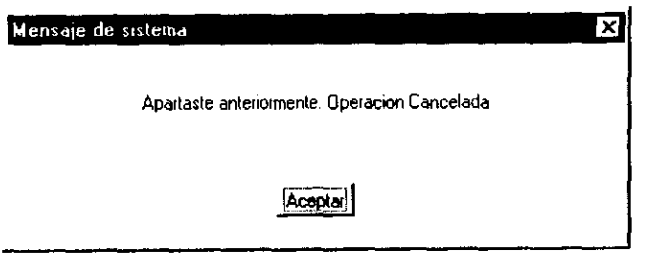


Figura 7-5

El botón con la etiqueta "Ver Estado Actual" muestra el estado actual de las computadoras en tiempo real. Este botón es útil si el usuario solicitó el estado de las computadoras en un intervalo de tiempo definido por él ya que el estado actual del laboratorio no se muestra hasta que el usuario selecciona este botón.

Los botones con las etiquetas "*¿Olvidaste tu Apartado?*", "*Cancelar Apartado*" y "*Salir*" no requieren que se introduzca un dato. El primero muestra la ventana de alerta con un mensaje que indica el login del usuario, el número de la computadora que apartó previamente, la hora de inicio y la hora de término de su apartado, si es que ese usuario tiene un apartado previo ya que en caso contrario muestra un mensaje de que no puede recordar el apartado ya que ese usuario no cuenta con él (Figura 7-6). El segundo cancela el apartado realizado previamente y muestra la ventana de alerta anunciando que la cancelación se realizó de manera exitosa o que no fue posible llevarla a cabo (Figura 7-7).

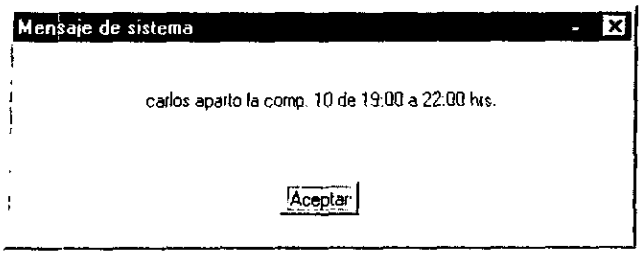


Figura 7-6

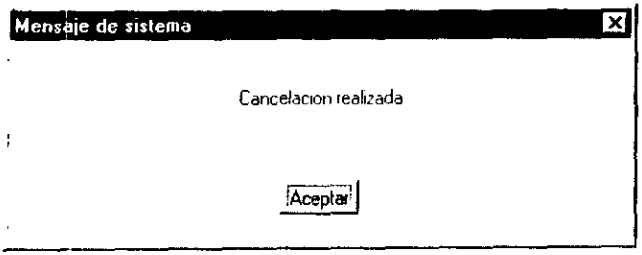


Figura 7-7

El tercer botón termina la sesión y el navegador carga el archivo Logout.html. En ese momento, el servidor recibe la petición de salida y termina la conexión:

```
Ha salido un usuario
Se desconecto el cliente 200.33.175.138!
```

Por ejemplo, en el caso mostrado aquí, el usuario no pudo realizar un apartado debido a que ya cuenta con un apartado previo. Lo que puede realizar es consultar su apartado previo y cancelarlo. En ese momento ya puede realizar un nuevo apartado en esta misma sesión sin necesidad de salir y volver a entrar al sistema (figura 7-8).

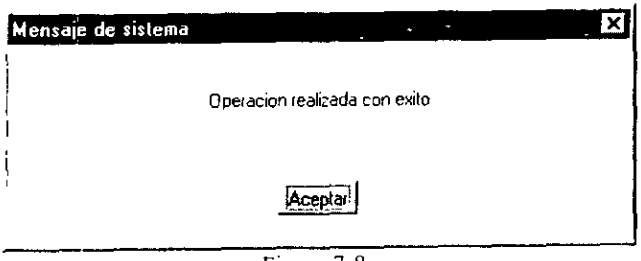


Figura 7-8

8. Conclusiones

El sistema actualmente ya se encuentra terminado y ha sido instalado en el laboratorio de cómputo de la DIE. Se tiene contemplado que en el sitio web donde reside, se reciban comentarios, sugerencias y posibles errores (bugs), que el sistema pudiera tener, vía email con la intención de crear un FAQ (Frecuently Asked Questions) y tener retroalimentación por parte de los usuarios así como colocar los archivos con código fuente para consulta de los usuarios interesados.

De este sistema, se concluye lo siguiente:

- El Paradigma Orientado a Objetos es, actualmente, la tendencia óptima para el análisis y diseño de sistemas.

- UML es un lenguaje de modelado de sistema que se aplica perfectamente en el análisis y diseño orientado a objetos tanto en la Ingeniería de Software como en otras ramas de Ingeniería.

- Java es un lenguaje de programación orientado a objetos que, por su naturaleza, es idóneo para el desarrollo de sistemas en ambientes distribuidos. Actualmente es muy utilizado para la realización de transacciones en ambientes de red.

- El World Wide Web (WWW) es actualmente un medio de comunicación muy útil para la realización de transacciones remotas seguras y efectivas.

Actualmente la tendencia en comunicaciones de red son los ambientes distribuidos y éstos se pueden implementar en aplicaciones realizadas en arquitecturas estándares como CORBA, DCOM y Java RMI (Remote Method Invocation) las cuales soportan componentes construidos en Java, C++, Pascal, ActiveX, PowerBuilder, etc.

9. Notas

Capítulo 3: El Análisis y el Diseño

[1] Kuhn, Thomas. 1962. *The Structure of Scientific Revolutions*. University of Chicago Press.

[2] Booch, G. 1994. *Object-Oriented Analysis and Design with Applications*. Second Edition. Redwood City, California: The Benjamin/Cummings Publishing Company, Inc., p. 17.

[3] Martin, James y Odell, James. 1992. *Análisis y Diseño Orientado a Objetos*. Prentice Hall, p. 18.

[4] Coad, P., y E. Yourdon. 1990. *Object-Oriented Analysis*. Prentice Hall, p. 1.

[5] Software Development Magazine. March 1998. *UML Applied: Nine tips to incorporating UML into your project*. Fuente: página web de Software Development Magazine. URL: http://www.sdmagazine.com/homepage.html.

Capítulo 4: Herramientas de desarrollo

[6] Campione, Mary y Walrath, Kathy. 1998. *The Java Tutorial: Object-Oriented Programming for the Internet*. Second Edition. Java Series Addison-Wesley Pub Co. http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

10. Bibliografía

Capítulo 3: El Análisis y el Diseño

Booch, Grady. 1994. *Object-Oriented Analysis and Design with Applications*. Second Edition. Redwood City, California: The Benjamin/Cummings Publishing Company, Inc.

Campione, Mary y Walrath, Kathy. 1998. *The Java Tutorial: Object-Oriented Programming for the Internet*. Second Edition. Java Series Addison-Wesley Pub Co. http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

Eriksson, Hans-Erik y Penker, Magnus 1998. *UML Toolkit*. Wiley Computer Publishing, John Wiley & Sons, Inc.

Martin, James y Odell, James. 1992. *Análisis y Diseño Orientado a Objetos*. Prentice Hall.

Capítulo 4: Herramientas de desarrollo

Arnold, Ken y Gosling, James. 1997. *El lenguaje de programación Java*. Java Series Addison Wesley Iberoamericana S. A./Domo Editores, S. L.

Byte Magazine, Enero 1997, Vol. 22 Número 1. *Java Security and Type Safety*.

Byte Magazine, Mayo 1998, Vol. 23 Número 5. *How to soup up Java*.

Campione, Mary y Walrath, Kathy 1998. *The Java Tutorial: Object-Oriented Programming for the Internet*. Second Edition. Java Series Addison-Wesley Pub Co. http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

Computer Magazine IEEE, Octubre 1995, Vol. 28 Número 10. *Object-Oriented Technology*.

December, John. 1995. *Presenting Java*. Sams Net Publishing

Capítulo 5: Propuesta de solución

Campione, Mary y Walrath, Kathy 1998. *The Java Tutorial: Object-Oriented Programming for the Internet*. Second Edition. Java Series Addison-Wesley Pub Co.
http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

Froufe, Agustín. 1996. *Tutorial de Java*.
http://www.fie.us.es/info/internet/JAVA/index.html

Hobbs, Ashton. 1998. *Aprendiendo programación para bases de datos con JDBC en 21 días*. Prentice Hall Hispanoamericana, S. A.

Lemay, Laura y Perkins Charles L. 1998. *Aprendiendo Java 1.1 en 21 días*. Segunda Edición. Prentice Hall Hispanoamericana, S. A.

Capítulo 6: Programación del sistema

Arnold, Ken y Gosling, James. 1997. *El lenguaje de programación Java*. Java Series Addison Wesley Iberoamericana S. A./Domo Editores, S. L.

Campione, Mary y Walrath, Kathy 1998. *The Java Tutorial: Object-Oriented Programming for the Internet*. Second Edition. Java Series Addison-Wesley Pub Co.
http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

Froufe, Agustín. 1996. *Tutorial de Java*.
http://www.fie.us.es/info/internet/JAVA/index.html

Java Platform 1.1 API Specification
http://java.sun.com/products/jdk1.1/products/api/index.html

Lemay, Laura y Perkins Charles L. 1998. *Aprendiendo Java 1.1 en 21 días*. Segunda Edición. Prentice Hall Hispanoamericana, S. A.

Naughton, Patrick. 1996. *Manual de Java*. Mc Graw-Hill. España