Capítulo II. Marco Teórico

El SICAAV será desarrollado en el lenguaje de programación Java, con el paradigma de programación orientada a objetos; se utilizará la arquitectura cliente-servidor y una base de datos relacional. Es por ello que este capítulo se referirá a cada uno de estos temas. Sin embargo no se debe olvidar que una parte esencial en el desarrollo de aplicaciones es la planeación, así como el periodo de pruebas, por lo que se describirá también el ciclo de vida del software.

2.1 Ingeniería de Software

Según Brooks la etapa de planeación del software representa un tercio del tiempo dedicado al desarrollo de un proyecto, solo una sexta parte a la codificación y la mitad del tiempo para las pruebas de componentes y de sistema respectivamente [Weitzenfeld, 2005]. Él también afirma que es la planeación, junto con sus etapas de levantamiento de requerimientos, análisis y diseño la parte esencial del desarrollo de software, mientras que la codificación y las pruebas son aspectos secundarios.

De acuerdo a Weitzenfeld la arquitectura del software debe ser “Administración de transacciones” debido a que tendremos interacción con bases de datos de manera concurrente y distribuida [Weitzenfeld, 2005]. Para el desarrollo de aplicaciones el mismo autor menciona que el ciclo de vida del software es el que sigue:

- Requisitos
- Análisis
- Diseño

- Implementación
- Integración
- Pruebas
- Documentación
- Mantenimiento

Requisitos

En esta etapa se especifican las necesidades del sistema a desarrollar. La especificación de requisitos sirve como base para la negociación entre los desarrolladores y los clientes del sistema y al mismo tiempo determinará el plan del desarrollo del software.

La parte crucial en esta etapa es cuando el experto explica al desarrollador los conceptos que el sistema debe manejar, pues muchas veces el experto en cuestión no sabe articular sus conocimientos; por lo que varias reuniones, explicaciones por escrito etc. serán necesarias.

Análisis

Una vez que se cuenta con los requisitos del sistema se tienen las herramientas para elegir la estructura del sistema. Ésta debe ser lógica, estable y extensible, el análisis se enfoca a qué es lo que debe de hacer el sistema. Se deben identificar los objetos y las relaciones que existen entre ellos.

Diseño

El diseño se refiere a las acciones necesarias para lograr el código final, es decir poner en práctica el sistema lógico y conceptual.

Implementación

En esta etapa se toma el diseño realizado previamente para llevar a cabo el código final del programa. La implementación supone ser una parte sencilla debido a que las decisiones importantes ya han sido tomadas.

Integración

Esta es una parte importante en el ciclo de vida, pues debido a que el sistema debe ser extensible, se debe tener la posibilidad de desarrollar módulos de manera independiente para posteriormente unirlos en el sistema principal.

Pruebas

Mediante estas pruebas se puede revisar la calidad del sistema. Principalmente las pruebas pueden ser de dos tipos, de validación y de verificación. La primera es para comprobar que la aplicación realice lo que el cliente desea y la verificación consiste en comprobar que el sistema funciona correctamente de acuerdo a lo planeado.

Documentación

Es importante que los manuales se realicen cuando se está elaborando el sistema. Las personas meta deben ser usuarios del sistema así como desarrolladores de posibles mejoras en el futuro.

Mantenimiento

Al tener la primera versión del sistema se debe iniciar con su mantenimiento.

2.2 Paradigma de Programación Orientada a Objetos

En la programación estructurada tanto los datos como las funciones eran globales, por lo que los programadores debían de trabajar pensando como máquinas y los cambios requerían de la revisión de todo el programa para hacer las correspondientes modificaciones.

En cambio la programación orientada a objetos representa un nivel de abstracción más alto pues se busca representar la realidad en cada uno de los objetos y los cambios en los datos deberían afectar únicamente al objeto que pertenecen y al grupo de funciones que éste contiene.

Las características que hacen de la programación orientada a objetos mejorar la calidad de los programas son: abstracción, modularidad, extensibilidad y reutilización.

Abstracción. En este tipo de programación la representación de los datos es mucho más cercana a la realidad.

Modularidad. Esta característica permite dividir un sistema, la programación orientada a objetos permite una mayor modularidad.

Extensibilidad. Los sistemas en general necesitan modificaciones, esta propiedad reduce los cambios a dos tipos, los que son propios del objeto y repercuten en éste y los que son externos a él y repercuten en el resto del sistema.

Reutilización. Gracias a la modularidad la reutilización de código es mucho mayor, la dificultad radica en el diseño de componentes genéricos y sencillos que puedan usarse en varias partes de la aplicación.

Clases y objetos

Los conceptos más importantes en la programación orientada a objetos son las clases y los objetos. Los objetos se forman por datos y operaciones que pueden modificar estos datos. En un programa se espera tener varios objetos del mismo tipo, por ejemplo si se maneja un programa para cuentas bancarias podríamos tener objetos como cuenta, cliente, ATM etc. y para el objeto cliente podríamos tener al Sr. Rodríguez y la Sra. López.

La clase por su parte es algo así como un molde para crear objetos. El objeto es la instancia de la clase.

Herencia y Polimorfismo

El polimorfismo se aplica a una sola variable que se refiere a objetos de diferentes clases [Wu, 2001]. Esto significa que al crear un objeto de una clase puede referirse a cualquier objeto de las clases que descienden de ella. Por ejemplo si tenemos la super clase Cliente con las clases descendientes ClientePremiun e ClienteGolden, la siguiente línea es correcta

Cliente arreglo= new ClientePremium();

Sin embargo no se puede hacer que una variable X se refiera a un objeto de la superclase o de las clases hermanas de X. Cuando se tiene un método con el mismo nombre en dos clases es polimórfico, debido a que una misma instrucción se refiere a dos métodos diferentes dependiendo de la clase.

Herencia y accesibilidad de miembros

Los variables y métodos de una clase pueden ser declarados como *public, prívate y protected. Public* indica que se puede acceder a este miembro desde cualquier método, los miembros *prívate* solo pueden será accesibles desde dentro de la clase y un miembro *protected* es accesible únicamente desde la misma clase o desde sus clases descendientes. Los constructores no son heredados por las subclases, por lo que se debe crear uno para cada clase.

Clases y métodos Abstractos

Un método abstracto se inicia con la palabra *abstract* y no tiene cuerpo. Una clase es abstracta si tiene un método de ese tipo o en caso de que no implemente un método abstracto heredado. Una superclase abstracta no puede ser instanciada, el objetivo de ser abstracta es que sus clases implementen sus métodos abstractos [Wu, 2001].

2.3 Arquitectura Cliente-Servidor

En esta arquitectura el servidor es normalmente una máquina potente que sirve de depósito de datos y funciona como SGBD; y el cliente es aquel que solicita algún servicio. De esta manera las transacciones se dividen en procesos independientes que cooperan entre si para intercambiar información, servicios o recursos. Las funciones generales de los clientes son:

- Manejo de la interfaz de usuario.
- Captura y validación de los datos de entrada.
- Generación de consultas e informes sobre las bases de datos.
- Por su parte los servidores realizan, entre otras, las siguientes funciones:

- Gestión de periféricos compartidos.
- Control de accesos concurrentes a bases de datos compartidas.
- Enlaces de comunicaciones con otras redes de área local o extensa.

Características físicas

Cuando se utiliza este modelo en aplicaciones de bases de datos se tiene muchas ventajas principalmente la optimización el uso de las aplicaciones *front-end* en las computadoras del cliente y aquellas ubicadas en el servidor. Así se mejora el acceso al servidor de base de datos a través de la red de área local y se cuenta con más usuarios para la misma información compartida.

Los servidores de bases de datos procesan peticiones y envíos de información, proporcionando únicamente la información que es requerida por las aplicaciones que se encuentran corriendo en el cliente. La computadora cliente es responsable únicamente de la ejecución de la aplicación cliente, de manejar la interacción con el usuario y de la generación de datos.

Características lógicas

Una de las principales aportaciones de esta arquitectura a los sistemas de información es la interfaz gráfica de usuario; gracias a ella se dispone de un manejo más fácil e intuitivo de las aplicaciones. En esta arquitectura los datos se presentan, editan y validan en la parte de la aplicación cliente.

En cuanto a los datos, cabe señalar que en la arquitectura cliente/servidor se evitan las duplicidades (copias y comparaciones de datos), teniendo siempre una imagen única y correcta de los mismos disponible en línea para su uso inmediato.

Todo esto tiene como fin que el usuario de un sistema de información soportado por una arquitectura cliente/servidor trabaje desde su estación de trabajo con distintos datos y aplicaciones, sin importarle dónde están o dónde se ejecuta cada uno de ellos [Khoshafian, Chan y Wong, 1993].

Las principales ventajas que ofrece esta tecnología son:

Mayor eficiencia en la división del trabajo. Tanto el cliente como el servidor se dedican a aquellas actividades para las que son mejores. El cliente puede ser utilizado para presentar interfaces gráficas y para manejar la interacción del usuario, mientras que el servidor se reserva para el procesamiento de altos volúmenes de datos con seguridad, integridad y control de concurrencia.

Estas aplicaciones generalmente pueden funcionar en pequeñas computadoras con un buen desempeño.

Debido a que los procesos pesados son llevados a cabo en el servidor, se pueden emplear PC’s de bajo rendimiento para ser utilizadas como clientes y tener acceso a la base de datos. Además el tráfico en la red es reducido pues solamente se envían los mensajes necesarios.

Los datos importantes pueden ser debidamente protegidos contra accesos inapropiados o pérdidas. Debido a que se tiene un DBMS centralizado los aspectos de seguridad, respaldo y recuperación de información son manejados de manera eficaz [Khoshafian, Chan y Wong, 1993].

2.4 Bases de datos relacionales

El diseño de la base de datos es un paso fundamental al desarrollar una aplicación cliente-servidor. Tiene que ver con disciplinas y lineamientos que se emplean en la construcción de bases de datos de alto desempeño y semánticamente correctas. Los tres componentes principales de este tipo de bases de datos son: estructura, restricciones de integridad e instancias o renglones de cada relación [Khoshafian, Chan y Wong, 1993].

En el mundo relacional, la estructura de una base de datos es una colección de relaciones o tablas. Cada tabla se organiza en columnas y renglones como se muestra en la figura 2.1.

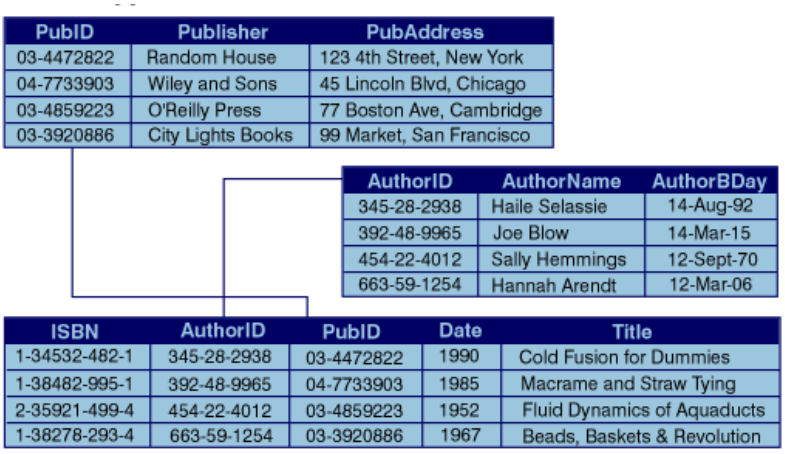

Figura 2.1 Modelo Relacional Hipotético

Relaciones o tablas

Una *relación* es definida como un conjunto de tuplas donde todas tienen los mismos atributos: *tabla*. En una base de datos relacional, todos los datos son almacenados en una *columna*, la que tiene determinado *tipo*, es decir tiene un conjunto de valores permisibles llamado dominio. En la práctica esto significa que los valores almacenados en una columna deben ser de mismo tipo de dato y de acuerdo a las mismas restricciones. Las tablas pueden ser tablas de base o vistas, las de base son aquellas que contienen los datos reales guardados en la base de datos; mientras que las vistas son el resultado de consultas llevadas a cabo sobre las tablas base. Las vistas no existen realmente en la base de datos.

Al conjunto de relaciones se le conoce como base de datos.

Diseño *Top-Down*

Existen diversos algoritmos y estrategias para el diseño de bases de datos relacionales. A continuación se explican los principales pasos para el diseño *top-down* [Khoshafian, Chan y Wong, 1993].

Paso 1. Identificar las entidades y relaciones del dominio del problema.

Paso 2. Combinar las vistas de los usuarios finales y generar el modelo conceptual.

Paso 3. Mapear el diagrama entidad-relación en el esquema relacional.

Paso 4. Implementar el diseño físico.

Paso1

Para identificar las entidades y relaciones del dominio del problema se debe entender al usuario y sus vistas, especificando los nodos y ligas en términos de entidades y relaciones. Mediante este proceso el diseñador modela el dominio del problema desde el punto de vista del usuario o tomando en cuenta las funciones del negocio y determina su correspondencia con el primer nivel externo de la arquitectura. En este proceso, las entidades, es decir las personas, los lugares, los objetos, los eventos y los papeles pueden ser especificados, junto con sus relaciones.

Identificar las entidades significa también identificar sus atributos. La información recopilada debe ser representada en diagramas entidad-relación.

Paso 2

La integración de las vistas de los usuarios en un modelo entidad-relación , construye la capa lógica del negocio y conforma el segundo nivel de la arquitectura.

Paso 3

Dependiendo del diagrama entidad-relación se tienen diferentes algoritmos para generar bases de datos normalizadas que tengan características de integridad y desempeño deseables.

Paso 4

En base a las especificaciones del dominio del problema y las diferentes aplicaciones de la empresa que estén siendo modeladas, se deben establecer clusters, indexaciones,

replicación de la información etc. Este esquema físico corresponde al tercer nivel de la arquitectura.

2.5 JDBC

La JDBC API es un conjunto de clases e interfaces que están escritas en el lenguaje java. Esta API hace fácil el envío de instrucciones en SQL para bases de datos relacionales; además hace posible el acceso a archivos que contienen datos tabulados; su valor radica en que las aplicaciones pueden accesar de manera virtual a cualquier fuente de información y en cualquier plataforma con una JVM [White, Fisher y Cattel, 1999]. Básicamente las tres acciones que realiza son:

1. Establecer una conexión con la fuente de información.
2. Enviar *queries* a la fuente de información.
3. Procesar los resultados.

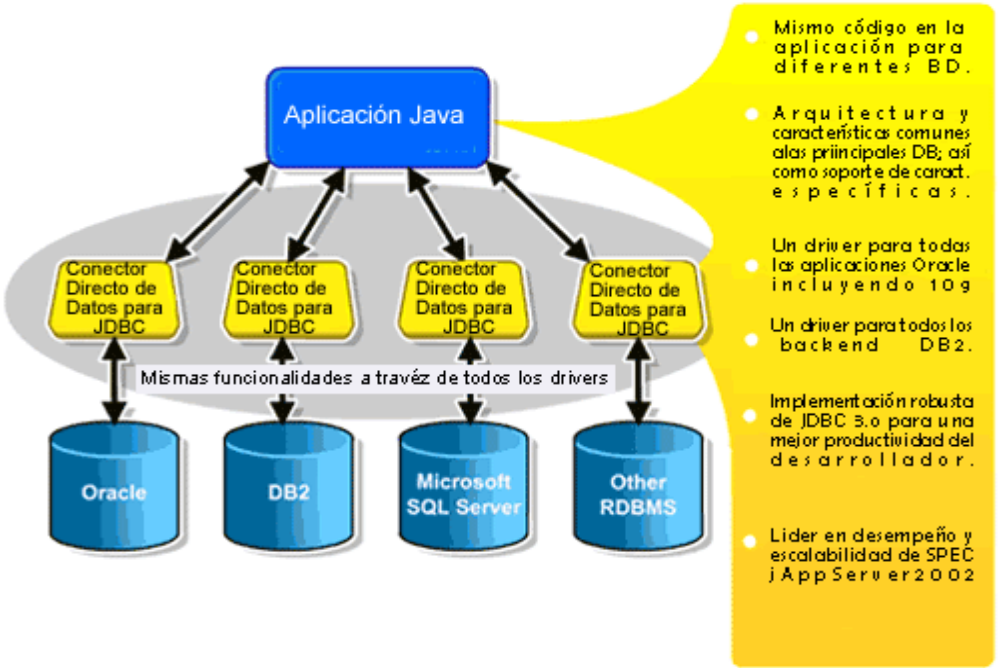

Figura 2.2. Diagrama de funcionalidad JDBC

Como se muestra en la figura 2.2 JDBC permite conectarse a cualquier clase de base de datos con el mismo código en la aplicación programada en java.

En un modelo de dos-tercios en el que la aplicación java se comunica directamente con la fuente de datos, se requiere un driver JDBC que se pueda comunicar con esta fuente de datos. Los comandos del usuario son entregados a la base de datos y los resultados de son enviados de regreso [White, Fisher y Cattel, 1999].

Para utilizar el JDBC se necesita cargar el driver indicado, esto se puede hacer con una sola línea de código.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Esta instrucción crea automáticamente una instancia del driver.

El segundo paso es establecer la conexión.

```
Connection con= DriverManager.getConnection(url,"myLogin","myPassword");
```

JDBC Statements

Un statement es lo que envía las consultas al DBMS. Lo que se tiene que hacer es crear un objeto Statement y ejecutarlo, enviando las consultas o inserciones al método *execute* apropiado. Para una consulta SELECT se debe usar el método executeQuery y la creación y modificación de tablas debe hacerse con el método executeUpdate.

```
Statement st= con.createStatement();
```

Después de crear el objeto se debe de ejecutar el método para llevar a cabo las consultas/ modificaciones.

St.executeUpdate(“Create table cofees”….” ;”);

Los resultados de las consultas realizadas son devueltas por java en un objeto ResultSet.

ResultSet rs = st.executeQuery(“select… “);

La variable rs es una lista con los resultados de la consulta realizada. Para obtener los atributos del resultado debemos de recorrer cada renglón con la función *next(),* una vez que aplicamos esta función el cursor apunta hacia el siguiente renglón y devuelve un valor *true* mientras que no esté en el último renglón. Por lo que podemos recorrer todo el ResultSet con un *while*. Para obtener el valor de cada columna del resultado utilizamos la función getXXX donde XXX puede ser String, Integer etc.

Mientras que el método executeQuery regresa un ResultSet con los resultados de la consulta el método executeUpdate regresa un int con el número de renglones afectados en la operación.

Transacciones

Cuando existen modificaciones en diferentes tablas o simplemente varios *update* que necesitamos realizar en conjunto, es decir que no se quiere que se realice una sin que se lleve a cabo la otra el conjunto de *queries* a realizar se conoce como transacciones.

Cuando una conexión es creada su modo *auto-commit* es verdadero, esto quiere decir que cada modificación es llevada a cabo inmediatamente, si se establece este modo como falso,

entonces podemos unir varias modificaciones en una transacción. Esto quiere decir que en sus valores originales la conexión lleva a cabo las modificaciones cuando los objetos resultSet o el número de renglones modificados ha sido entregado.

Con.setAutoCommit(false);

Al cambiar el modo a falso debemos de llamar al método commit de manera explícita para que las modificaciones y consultas se lleven a cabo.

RollBack

Cuando se usa el método *rollback* se aborta la transacción en cuestión y los valores que fueron modificados regresan a su valor anterior. Esta función debe implementarse cuando existe algún error en la ejecución de las modificaciones, dentro de un *catch{..}*

Estos son los conocimientos básicos que debe tenerse para acceder y modificar una base de datos desde java [White, Fisher y Cattel, 1999].

2.6 Conclusión

En este capítulo se ha hablado sobre los temas necesarios para implementar el SICAAV. En el siguiente capítulo se tratará el diseño de la aplicación.