Diseño e implementación de una interfaz abierta sobre el web para la ejecución de sentencias SQL sobre motores de bases de datos relacionales

Garzón, Gonzalo

Mantilla, Juan Carlos

Rosa Upegui, Margarita

Director.

Universidad Tecnológica De Bolívar

Maestría en Ciencias Computacionales

Cartagena de Indias

2006

INTRODUCCIÓN

Este proyecto se concibió como producto de una necesidad vivida en el ámbito empresarial y académico donde se manejan múltiples bases de datos y para cada una de ellas hay que capacitar al personal o al estudiante en las particularidades de las herramientas clientes de las mismas.

Las salas y horarios disponibles de las salas generalmente son limitadas y es muy beneficioso si el estudiante cuenta con una herramienta para realizar prácticas de SQL, el lenguaje estándar de las bases de datos a través del web para conectarse al DMBS con que cuente la universidad donde estudia desde su hogar u otro sitio donde tenga acceso al Web.

Los clientes de motores de bases de datos son generalmente costosos para el ámbito académico donde lo que interesa es la interacción con los DBMS en sus características más generales. Este proyecto pretende aportar al ámbito académico una herramienta básica para enseñanza del SQL estándar y las características nativas de los motores de bases de datos más accesibles en el mercado.

El producto es una aplicación cliente/servidor que funcionara con interfaz web (en la Intranet o desde Internet), realizada en Java, totalmente orientada a Objetos, que permitirá el acceso a una de múltiples bases de datos.

2. COMPILADORES Y HERRAMIENTAS GENERADORAS

La necesidad de establecer comunicación con dispositivos de cómputo para un creciente número de usuarios ha obligado a construir herramientas que permitan que esta comunicación se realice de manera más efectiva y con menor consumo de tiempo. Esto se puede apreciar desde finales de la década de los 50's, cuando con el advenimiento de computadoras comerciales surge también la necesidad de programarlas. Se diseñaron lenguajes como FORTRAN y COBOL que permiten realizar esta tarea de comunicación al establecer una relación entre los problemas de los usuarios y lo que las máquinas eran capaces de realizar.

Estos primeros lenguajes también vinieron acompañados de un nuevo término: Compilador. Se le atribuye a Grace Murray Hopper la acuñación de este término y se refería al trabajo que estaba detrás de la programación en aquellos tiempos, existía una biblioteca de programas constituida de un conjunto de rutinas, cada una de ellas probada individualmente; cuando se necesitaba un programa, se elegían las rutinas necesarias de esa biblioteca y se integraban para conformar el proceso que ejecutaría la computadora. Quién realizaba este trabajo de acopio de rutinas y de integración se le denominaba compilador, de ahí que los nuevos lenguajes tuviesen sus propios "compiladores" para la integración del proceso que programar representaba.

En estos días, el término aún se conserva aunque con un sentido ligeramente diferente al planteado por Hopper. Hoy en día, un compilador es un traductor que facilita la comunicación entre el programador y la máquina, por medio de un proceso de transformación.

Este proceso se puede dividir en las siguientes fases:

Análisis de Léxico, Análisis gramatical o sintáctico, Análisis semántico y de verificación de tipos, Generación de código intermedio, Optimización de código intermedio, Generación de código final. Construir un compilador puede ser una tarea ardua; en ocasiones lo es por la naturaleza del lenguaje y en otras, por el equipo que lo construye. Se requiere tener una gran precisión para coordinar acciones y obtener resultados rápidamente. Afortunadamente generadores de compiladores que aligeran el trabajo de construcción.

2.1 COMPONENTES DE UN COMPILADOR Y DEFINICIONES

2.1.1 Analizador de Léxico. Un analizador de léxico tiene como función principal el tomar secuencias de caracteres o símbolos del alfabeto del lenguaje y ubicarlas dentro de categorías, conocidas como unidades de léxico. Las unidades de léxico son empleadas por el analizador gramatical para determinar si lo escrito en el programa fuente es correcto o no gramaticalmente. Algunas de las unidades de léxico no son empleadas por el analizador gramatical sino que son descartadas o filtradas. Tal es el caso de los comentarios, que documentan el programa pero que no tienen un uso gramatical, o los espacios en blanco, que sirven para dar legibilidad a lo escrito[1].

2.1.2 Análisis Gramatical. Una de las situaciones más complejas en el desarrollo de un lenguaje es el establecimiento de las reglas gramaticales que éste debe cumplir. La bondad o tortura que puede ser el emplear un lenguaje determinado se debe, en parte, a la facilidad de describir situaciones por medio del lenguaje que representen un conjunto de acciones específicas. Contrariamente a los lenguajes naturales, los lenguajes computacionales son más restrictivos en cuanto a su capacidad de expresión ya que, a diferencia de un idioma, una computadora no debe entender más que de una sola manera lo que un programador escribe. Se dice que las gramáticas que soportan a los lenguajes de cómputo deben ser independientes al contexto o forma en que se emiten sus enunciados (no importa quien lo emita, en qué momento o con qué entonación: debe significar siempre lo mismo).

[1] AHO, Alfred V.; SETHI, Ravi; ULLMAAN, Jeffry D. (Agosto 1977) Compiladores Princpios, tecnicas y herramientas. Pg. 40

Como diseñadores de lenguajes se debe con seguir el paradigma del lenguaje que se usa más o con el que se siente mayor agrado; la forma de describir una gramática está dada por la notación BNF, que trata primordialmente la construcción de gramáticas libres al contexto. Esta notación está basada en trabajos de Noam Chomsky y se ha usado extensivamente para declarar las gramáticas de la mayoría de los lenguajes que se emplea hoy en día[2].

Gramáticas Libres al Contexto. Formalmente una gramática libre al contexto esta constituida por:

- Un conjunto de tokens conocidos como símbolos terminales.
- Un conjunto de no terminales.
- Un conjunto de producciones donde cada producción consiste de un no terminal (llamado el lado izquierdo de la producción), una flecha, y una secuencia de tokens y no terminales. (llamado el lado derecho de la producción).
- La designación de uno de los no terminales como el símbolo inicial (o axioma de la gramática)[3].

Analizadores Gramaticales. Al construir la gramática que guiará a un lenguaje computacional, se debe construir el analizador gramatical que determinará si se está cumpliendo o no las reglas que el lenguaje posee. Dependiendo de la forma en que realiza el análisis, existen dos tipos de analizadores gramaticales:

- Analizadores ascendentes
- Analizadores descendentes

Un analizador ascendente tiene como objetivo construir el axioma de la gramática a partir de los elementos de léxico que recibe; se denomina ascendente por que es a partir de los elementos más simples que espera conformar el nivel más alto en la gramática: el axioma.

[2] AHO, Alfred V.; SETHI, Ravi; ULLMAAN, Jeffry D. (Agosto 1977) Compiladores Princpios, tecnicas y herramientas. Pg. 85
[3] AHO, Alfred V.; SETHI, Ravi; ULLMAAN, Jeffry D. (Agosto 1977) Compiladores Princpios, tecnicas y herramientas. Pg. 87

Por el contrario, un analizador gramatical descendente busca descomponer el axioma de la gramática en los elementos que recibirá del analizador de léxico (y que representan al programa que está siendo revisado). Este método es el menos empleado hoy día para construir un analizador gramatical ya que requiere que la gramática sufra varias transformaciones antes de poderse emplear para realizar el análisis[4].

2.2 HERRAMIENTAS GENERADORAS ACTUALES

En la actualidad no es posible pensar en la realización computacional de un procesador de lenguajes, como por ejemplo, compiladores, intérpretes, etc., sin utilizar una herramienta instrumental, tal como un generador de compiladores. Este tipo de herramientas existe hace años, y han ido evolucionando junto con la propia evolución de la Ciencia de la Computación, encontrando en la actualidad muchos y diferentes tipos de generadores de compiladores, que en general corresponden con los entornos de desarrollo de aplicaciones y con los paradigmas de la programación.

Entre los generadores de compiladores más conocidos y más utilizados podemos encontrar los siguientes:

- Lex / Yacc
- Bison
- Visual Parse ++
- JavaCC
- Antlr

2.3 HERRAMIENTA GENERADORA DE PARSER USADA: ANTLR

La herramienta generadora de Parser usada es ANTLR. Sus análisis lexicográfico y sintáctico se basan en gramáticas LL(k), y para el análisis semántico se usan las

[4] AHO, Alfred V.; SETHI, Ravi; ULLMAAN, Jeffry D. (Agosto 1977) Compiladores Princpios, tecnicas y herramientas. Pg. 88

gramáticas L-atributadas. Incorpora los conceptos de símbolos ignorables, que pueden declararse locales a una producción, y flujos de símbolos. Incorpora la operación de negación, pudiéndose hacer relativa a los símbolos de la gramática o al vocabulario de caracteres, pues ANTLR reconoce el conjunto de caracteres Unicode.

Durante el análisis lexicográfico, la cadena de caracteres asociada a un símbolo terminal puede ser transformada, logrando algo equivalente a los estados léxicos MORE de JavaCC.

ANTLR usa la correspondencia entre gramática y clase, excluyendo el anidamiento de declaraciones de clases. En una especificación puede darse un número arbitrario de definiciones de analizadores, que pueden ser lexicográficos, sintácticos o de árboles --- desde su punto de vista, los árboles son también un flujo de símbolos, sólo que en dos dimensiones.

Todas las definiciones en ANTLR son clases, existiendo tres clases raíces: Lexer, Parser, y TreeParser. El tipo de analizador que se defina dependerá de cuál de estas tres clases se haga heredar (transitivamente) la definición. El conjunto de superclases disponibles para una clase dada, está constituido por las clases definidas en el mismo fichero y en ficheros especificados con la opción -glib. La implementación de la herencia tiene una pequeña incongruencia: teniendo una gramática G que genere una clase CG, para ninguna gramática que herede de G se cumplirá que la clase generada a partir de ella sea heredera de CG.

Los componentes de una definición de ANTLR son las opciones, las reglas y las acciones semánticas. Al heredar puede redefinirse cualquiera de estos componentes de la definición base, o adicionar nuevos. En el caso de las producciones, si se desea adicionar alguna a la definición de un símbolo, deben reescribirse todas las asociadas al mismo. Con ProGrammar [Nor00] la extensión de la definición de un símbolo se hace de un modo en el cual no surge esta inconveniencia.

Los análisis lexicográfico y sintáctico se implementan usando el método recursivo-descendente con "lookahead" local y predicados sintácticos y semánticos. Los últimos los

clasifica en validadores y desambiguadores, los cuales, permiten instrumentar estrategias de tratamiento de errores semánticos más allá del contexto de la regla donde se generó. La evaluación de los atributos se hace a la par del análisis sintáctico, y se instrumentan los atributos sintetizados y heredados usando los parámetros de las funciones y sus valores de retorno.

El código generado puede obtenerse en uno de los lenguajes de programación siguientes: Java, C++, o Sather. Generadores hacia otros lenguajes pueden adicionarse sin necesidad de recompilar ANTLR, pues las clases que implementan la generación de código se localizan usando la API de Reflexión de Java.

Existe una herramienta llamada ParseView [Par b], con una interfaz gráfica para la depuración de errores del analizador generado, con la opción de la línea de comandos **-debug**, y que será llamada automáticamente por el analizador al ejecutarse. De forma equivalente pueden usarse las opciones de la línea de comandos **-trace**, **-traceLexer**, **-traceParser**, y **-traceTreeWalker**, que harán que sean invocadas las funciones **traceIn** y **traceOut** por todas las reglas o, respectivamente, por las reglas de cada tipo de analizador.

ANTLR se construye a partir de una gramática definida en su propio lenguaje de especificación, consecuentemente con la arquitectura de procesamiento de lenguajes que promueve, y por tanto puede procesar una especificación sin importar cómo o dónde esté almacenada, siempre que exista una clase que implemente el flujo de símbolos del cual pueda leerla. De hecho, la herramienta por línea de comandos (antlr.Tool) está implementada de esa forma.

3. SISTEMAS DE GESTION DE BASES DE DATOS (DBMS)

Un sistema de gestión de base de datos (DBMS D*atabase Management System*) consisten en una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. El objetivo primordial de un DBMS es proporcionar un entorno que sea a la vez conveniente y eficiente para ser utilizado al extraer y almacenar información de la base de datos.

Los sistemas de bases de datos están diseñados para gestionar grandes bloques de información. La gestión de datos implica tanto la definición de estructuras para el almacenamiento de información como la provisión de mecanismos para gestión de la información. Además, los sistemas de bases de datos deben mantener la seguridad de la información almacenada, pese a caídas del sistema o intentos de accesos no autorizados. Si los datos van a ser compartidos por varios usuarios, el sistema debe evitar posibles resultados anómalos. La importancia de la información en la mayoría de las organizaciones, y por tanto el valor de la base de datos, ha llevado al desarrollo de una gran cantidad de conceptos y técnicas para la gestión eficiente de los datos.

3.1 CARACTERÍSTICAS DE UN DBMS

3.1.1 Abstracción de Datos. Un objetivo principal de un sistema de base de datos es proporcionar a los usuarios una visión abstracta de los datos. Esto es, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos. Esto se logra definiendo tres niveles de abstracción en los que se puede ver la base de datos: el nivel físico, el nivel conceptual y el nivel de visión.

El *nivel físico* es el nivel mas bajo de abstracción, describe cómo se almacenan realmente los datos. En el nivel físico, se describen en detalle las estructuras de datos complejas del nivel bajo.

El *nivel conceptual* es el siguiente nivel más alto de abstracción, describe qué datos son realmente almacenados en la base de datos y las relaciones que existen entre los datos. El nivel conceptual de abstracción lo usan los administradores de bases de datos, quienes decidir qué información se va a guardar en la base de datos.

El *nivel de visión* es el nivel más alto de abstracción, describe sólo parte de la base de datos completa. Muchos usuarios del sistema de base de datos no se interesan por toda la información. En cambio, dichos usuarios sólo necesitan una parte de la base de datos. Para simplificar su interacción con el sistema se define el nivel de abstracción de visión. El sistema puede proporcionar muchas visiones para la misma base de datos[5].

3.1.2 Modelos de Datos. Para describir la estructura de una base de datos es necesario definir el concepto de modelo de datos, una colección de herramientas conceptuales para describir datos, relaciones entre ellos, semántica asociada a los datos y restricciones de consistencia. Los diversos modelos de datos que se han propuesto se dividen en tres grupos: modelos lógicos basados en objetos, modelos lógicos basados en registros y modelos físicos de datos[6].

Los ***modelos lógicos basados en objetos*** se usan para describir datos en los niveles conceptual y de visión. Se caracterizan por el hecho de que proporcionan capacidad de estructuración bastante flexible y permiten especificar restricciones de datos explícitamente. Hay muchos modelos diferentes, y es probable que aparezcan más. Algunos de los más extensamente conocidos son:

- El modelo entidad-relación
- El modelo orientado a objetos
- El modelo binario
- El modelo semánticos de datos
- El modelo infológico
- El modelo funcional de datos.

[5] KORTH, Henry; SILBERSCHATZ, Abraham (1993). Fundamentos de Base de Datos. Ed. McGraw-Hill. p. 4
[6] KORTH, Henry; SILBERSCHATZ, Abraham (1993). Fundamentos de Base de Datos. Ed. McGraw-Hill. p. 6

Los ***modelos lógicos basados en registros*** se utilizan para describir datos en los modelos conceptual y físico. A diferencia de los modelos de datos basados en objetos, se usan para especificar la estructura lógica global de la base de datos y para proporcionar una descripción a nivel más alto de la implementación.

Los modelos basados en registros se llaman así porque la base de datos está estructurada en registros de formato fijo de varios tipos. Cada tipo de registro define un número fijo de campos, o atributos, y cada campo normalmente es de longitud fija.

Los tres modelos de datos más ampliamente aceptados son los modelos relacional, de red y jerárquico.

El *modelo relacional* representa los datos y las relaciones entre los datos mediante una colección de tablas, cada una de las cuales tiene un número de columnas con nombres únicos.

En el *modelo de red* los datos se representan mediante colecciones de registros y las relaciones entre los datos se representan mediante enlaces, los cuales pueden verse como punteros. Los registros en la base de datos se organizan como colecciones de grafos arbitrarios.

El *modelo jerárquico* es similar al modelo de red en el sentido de que los datos y las relaciones entre los datos se representan mediante registros y enlaces, respectivamente. Se diferencia del modelo de red en que los registros están organizados como colecciones de árboles en vez de grafos arbitrarios.

Los ***modelos físicos de datos*** se usan para describir datos en el nivel más bajo. A diferencia de los modelos lógicos de datos, hay muy pocos modelos físicos de datos en uso. Dos de los más ampliamente conocidos son:

- Modelo unificador
- Memoria de elementos

3.1.3 Instancias y Esquemas. La colección de información almacenada en la base de datos, en un determinado momento en el tiempo, se llama una instancia de la base de datos. El diseño global de la base de datos se llama esquema de la base de datos. Los esquemas se cambian muy raras veces, o nunca.

Los sistemas de bases de datos tienen varios esquemas. En el nivel más bajo está el esquema físico; en el nivel intermedio, el esquema conceptual; en el nivel más alto, un subesquema. En general, los sistemas de bases de datos soportan un esquema físico, un esquema conceptual y varios subesquemas[7].

3.1.4 Independencia de Datos. La capacidad de modificar una definición de un esquema en un nivel sin afectar la definición de un esquema en el nivel superior siguiente se llama independencia de datos. Hay dos niveles de independencia de datos[8]:

- *Independencia física de datos* es la capacidad de modificar el esquema físico sin provocar que se vuelvan a escribir los programas de aplicación.
- *Independencia lógica de datos* es la capacidad de modificar el esquema conceptual sin provocar que se vuelvan a escribir los programas de aplicación.

3.1.5 Lenguaje de Definición de Datos (DDL). Un esquema de base de datos se especifica por medio de un conjunto de definiciones que se expresan mediante un lenguaje especial llamado lenguaje de definición de datos (*data definition language* (DDL)). El resultado de la compilación de sentencias de DDL es un conjunto de tablas las cuales se almacenan en un archivo especial llamado *diccionario de datos* (o directorio).

Un diccionario de datos es un archivo que contiene *metadatos*, es decir, “datos sobre datos”[9].

3.1.6 Lenguaje de Manipulación de Datos (DML). Es un lenguaje que permite a los usuarios acceder o manipular los datos. Existen básicamente dos tipos: *DML Procedimentales* que requieren que el usuario especifique qué datos se necesitan y cómo

[7] KORTH, Henry; SILBERSCHATZ, Abraham (1993). Fundamentos de Base de Datos. Ed. McGraw-Hill. p. 12
[8] KORTH, Henry; SILBERSCHATZ, Abraham (1993). Fundamentos de Base de Datos. Ed. McGraw-Hill. p. 13
[9] KORTH, Henry; SILBERSCHATZ, Abraham (1993). Fundamentos de Base de Datos. Ed. McGraw-Hill. p. 13

obtenerlos, y *DML No Procedimentales* que requieren que el usuario especifique qué datos se necesitan sin especificar cómo obtenerlos.

Una *consulta* es una sentencia que solicita la recuperación de información. El trozo de un DML que implica recuperación de información se llama *lenguaje de consultas*. Aunque técnicamente es incorrecto, suelen utilizarse los términos *lenguaje de consultas* y *lenguaje de manipulación de datos* como sinónimos[10].

3.1.7 Gestor de Base de Datos. Es un modulo de programa que proporciona el interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y las consultas que se hacen al sistema. El gestor de base de datos es responsable de interaccionar con el gestor de archivos, de mantener la integridad, de garantizar la seguridad, de las copias de seguridad y recuperación, y del control de concurrencias[11].

3.2. DBMS ACTUALES Y DBMS A LOS CUALES PUEDE CONECTARSE A HERRAMIENTA

En la actualidad existen diversos DBMS, los cuales tienen cada uno distintas prestaciones a nivel de rendimiento, rapidez y efectividad en el manejo de la información almacenada.

Teniendo en cuenta la rapidez en las consultas, se pueden mencionar algunos como MySQL o PointBase, que sacrifican funcionalidad o soporte del Standard ANSI SQL92, por ofrecer rapidez en las búsquedas, lo cual los hace ideales para aplicaciones en las cuales la búsqueda de información es lo prioritario como por ejemplo, las aplicaciones Web, en las cuales se despliega información contenida en bases de datos.

Brillan por su robustés DBMS como PostgreSQL, Informix, MSSQL Server u Oracle, siendo los dos últimos los DBMS más populares en el mercado y los cuales ofrecen un conjunto de herramientas para una fácil administración de la gran cantidad de funciones y características que poseen. Estos motores son ideales para el almacenamiento de

[10] KORTH, Henry; SILBERSCHATZ, Abraham (1993). Fundamentos de Base de Datos. Ed. McGraw-Hill. p. 14
[11] KORTH, Henry; SILBERSCHATZ, Abraham (1993). Fundamentos de Base de Datos. Ed. McGraw-Hill. p. 15

grandes cantidades de datos, en sistemas corporativos muy grandes que requieren la confiabilidad de un DBMS de alto rendimiento.

La aplicación que tiene la capacidad de conectarse a los siguientes DBMS's:

- MySQL
- Microsoft SQL Server 2000
- Oracle
- y cualquier otro motor de bases de datos para el que se tenga el driver respectivo y se aplique la función "Agregar DBMS" para adicionar otro motor de Base de datos dentro de la opción de Admón. de la Aplicación.

3.3 ODBC, JDBC Y OLEDB

ODBC (por sus siglas en inglés Open DataBase Connectivity), es un modelo de conexión a bases de datos, desarrollado por Microsoft, y que se ha convertido en un estándar para la conectividad con bases de datos relacionales en el momento actual. Existen varias versiones de esta API, con la cual los desarrolladores proveen de controladores ODBC para los distintos motores de bases de datos, cuidando de cumplir con la especificación ODBC.

Este modelo de acceso a datos se ha venido utilizando desde hace mucho tiempo en el ámbito computacional, pero a medida que han crecido las versiones y los distintos motores de bases de datos, se ha vuelto complicado manejar tantos controladores para ellos.

SUN Microsystems, lanzó entonces una interfaz estándar, JDBC (Java DataBase Connectivity), para conectarse desde Java hacia bases de datos relacionales. El estándar JDBC fue definido por SUN Microsystems, permitiéndoles a los proveedores individuales implementar y extender el estándar con sus propios controladores JDBC, reemplazando las APIs ya existentes, como ODBC, con la intención de obviar los problemas que presenta el uso desde Java de estas APIs, que suelen ser de muy bajo nivel y utilizar características no soportadas directamente por Java, como punteros, etc. Aunque el nivel

de abstracción al que trabaja JDBC es alto en comparación, por ejemplo, con ODBC, la intención de SUN es que sea la base de partida para crear librerías de más alto nivel, en las que incluso el hecho de que se use SQL para acceder a la información sea invisible.

JDBC está basado en X/Open SQL Call Level Interface y cumple con el estándar SQL ANSI92.

OLE DB: Bajo este nombre, Microsoft denomina a su nueva estrategia en el mercado de de las base de datos. Básicamente UDA es una tecnología basada en el Component Object Model (COM) y que implementa una serie de interfases de programación y acceso a bases de datos.

OLE DB es un estándar multiplataforma de Microsoft de acceso a datos, basado en la tecnología COM. Mientras ODBC usa aplicaciones estáticas para acceder a fuentes de datos, además de estar limitado al SQL para el traslado de éstos; OLE DB es rápido, simplifica el desarrollo de aplicaciones y lo acelera usando ADO: la interfaz sencilla y rápida de desarrollo para OLE DB. Es sencillo de aprender para los programadores y puede ser utilizado en la mayoría de los entornos populares de programación.

OLE DB se basa, como se dijo, en tecnología COM y ofrece una serie de requisitos mínimos al desarrollador en forma de un alto nivel de rendimiento tanto en sistemas locales como remotos, soporte para seguridad, sistema transaccional, e integración con sistemas más antiguos y otras plataformas.

La potencia real de OLE DB es que permite al acceso a muchos tipos de datos distintos. Anteriormente a este sistema, la solución más universal en plataformas Windows siempre ha sido el estándar ODBC. Mediante ODBC se puede acceder a cualquier tipo de base de datos siempre que dicha base de datos ofreciese un modelo relacional compatible con el lenguaje de consulta SQL.

3.4. LENGUAJE SQL

Las bases de datos relacionales son aplicaciones que controlan los datos en una base de datos incluyendo organización, almacenamiento, recuperación, seguridad e integridad de la información. Tiene un lenguaje de manipulación de datos para procesar las consultas denominado SQL.

El nombre "SQL" es una abreviatura de Structured Query Languaje (Lenguaje de consultas estructurado). El SQL es a la vez un lenguaje fácil de aprender y una herramienta completa para gestionar datos. Las peticiones sobre los datos se expresan mediante sentencias, que deben escribirse de acuerdo con unas reglas sintácticas y semánticas de este lenguaje.

La importancia del aprendizaje del SQL es que no es puntual a una Base de datos específica sino que la mayoría de los DBMS actuales aplican el lenguaje SQL específicamente la del estándar SLQ92 ya que es un lenguaje estándar por haberse visto consolidado por el Instituto Americano de Normas (ANSI) y por la Organización de Estándares Internacional (ISO).

Una nueva versión del Standard para SQL (SQL99) fue desarrollada para incluir funciones avanzadas al lenguaje; pero el corazón del SQL, tal como adicionar, leer y modificar datos son las mismas. Al comienzo del 2001 ningún vendedor había aplicado las funciones del nuevo estándar. Lo más importante para nosotros es que las instrucciones del estándar SQL-ANSI92 son de igual importancia académica a la fecha y son todavía la principal guía en la mayoría de los cursos de introducción al SQL para todos los motores de Bases de datos relacionales.

Existen básicamente tres tipos de comandos SQL:

- Los DDL (Data definition Language) que permiten crear (definir) nuevas bases de datos, campos e índices. Y borrarlos (Instrucciones create, alter, y drop)
- Los DML (Data Manipulation Language) que permiten generar consultas para ordenar, filtrar, borrar y extraer datos de la base de datos. (insert, delete, update, select, commit, rollback)

Nuestro sistema (MULTI-SQL) solo valida localmente la sintaxis del DML ingresado por el estudiante en el recuadro de texto. En caso de tratarse otro tipo de instrucciones (DDL o DCL) el comando debe ejecutarse desde la misma interfaz para ejecutarla directamente en el motor de base de datos al cual se esta conectado. Lo cual permite que si el usuario no desea usar el validador local puede trabajar todo el tiempo con la sintaxis de un motor específico para los tres tipos de comandos. Si desea validar que toda su sintaxis sea del estándar ANSI92 del SQL deberá escoger la opción "VALIDAR" para revisar localmente la sintaxis de su instrucción. Esto agrega una versatilidad adicional al software que puede usarse para capacitar en SQL estándar o en el lenguaje particular asociado al sistema de gestión de bases de datos con que se cuenta.

Comandos DLL	
Comando	**Descripción**
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Tabla 1 Comandos DLL

Comandos DML	
Comando	**Descripción**
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados
DELETE	Utilizado para eliminar registros de una tabla de una base de datos

Tabla 2. Comandos DML

Cláusulas: Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

Tabla 3. Descripción de cláusulas

Operadores Lógicos:

Operador	Uso
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Tabla 4. Uso de los Operadores lógicos

Operadores de Comparación:

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos

Tabla 5. Uso de los operadores de Comparación

Funciones de Agregación: Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Función	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

Tabla 6. Descripción de las funciones de agregado

4. DISEÑO DE APLICACIONES MVC

La arquitectura MVC (*Model/View/Controller*) fue introducida como parte de la versión Smalltalk-80 del lenguaje de programación Smalltalk. Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales son que el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas. Para profundizar en este tema se hará antes una introducción al modelo Cliente Servidor.

4.1 ARQUITECTURA CLIENTE/SERVIDOR

Es la tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo a través de la organización, en múltiples plataformas. El modelo soporta un medio ambiente distribuido en el cual los requerimientos de servicio hechos por estaciones de trabajo inteligentes o "clientes'', resultan en un trabajo realizado por otros computadores llamados servidores"

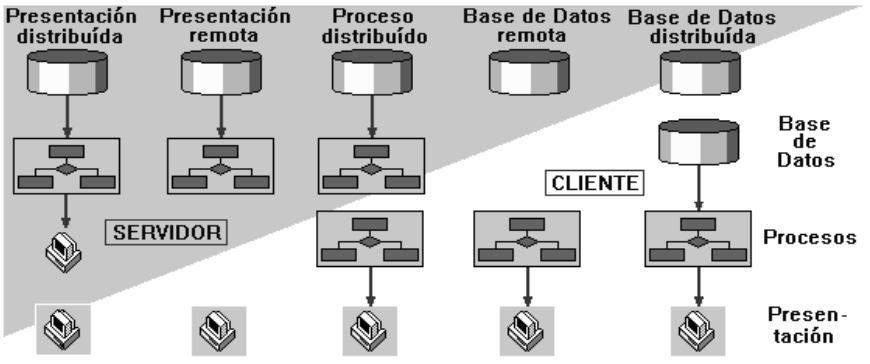

Figura 1. Arquitectura Cliente/Servidor

Entre las principales definiciones se tiene:

- Desde un punto de vista conceptual: Es un modelo para construir sistemas de información, que se sustenta en la idea de repartir el tratamiento de la información y los datos por todo el sistema informático, permitiendo mejorar el rendimiento del sistema global de información.
- En términos de arquitectura: Los distintos aspectos que caracterizan a una aplicación (proceso, almacenamiento, control y operaciones de entrada y salida de datos) en el sentido más amplio, están situados en más de un computador, los cuales se encuentran interconectados mediante una red de comunicaciones.

4.1.1 Estructura física. La idea principal consiste en aprovechar la potencia de los ordenadores personales para realizar sobre todo los servicios de presentación y según el nivel, algunos procesos o incluso algún acceso a datos locales. De esta forma se descarga al servidor de ciertas tareas para que pueda realizar otras más rápidamente.

También existe una plataforma de servidores que sustituye al ordenador central tradicional y que da servicio a los clientes autorizados. Incluso a veces el antiguo ordenador central se integra en dicha plataforma como un servidor más.

Estos servidores suelen estar especializados por funciones (seguridad, cálculo, bases de datos, comunicaciones, etc.), aunque, dependiendo de las dimensiones de la instalación se pueden reunir en un servidor una o varias de estas funciones.

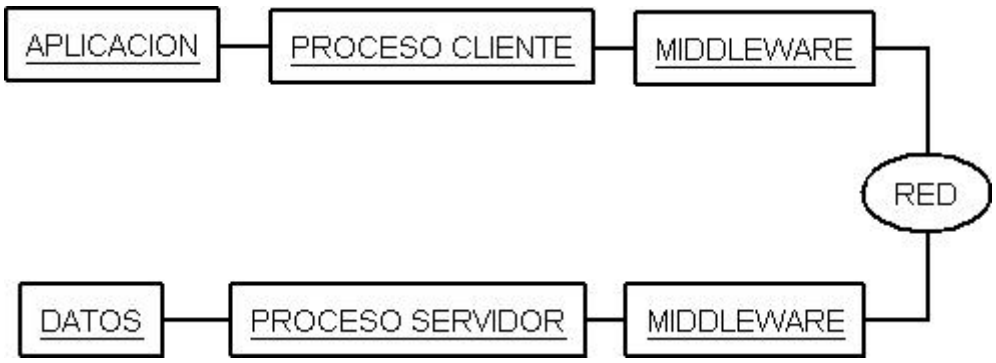

Figura 2. Estructura física de un modelo Cliente/Servidor

Las unidades de almacenamiento masivo en esta arquitectura se caracterizan por incorporar elementos de protección que evitan la pérdida de datos y permiten multitud de accesos simultáneos (alta velocidad, niveles RAID, etc.)
Para la comunicación de todos estos elementos se emplea un sistema de red que se encarga de transmitir la información entre clientes y servidores. Físicamente consiste en un cableado (coaxial, par trenzado, fibra óptica, etc.) o en conexiones mediante señales de radio o infrarrojas, dependiendo de que la red sea local (RAL), metropolitana (MAN) o de área extensa (WAN)

Para la comunicación de los procesos con la red se emplea un tipo de equipo lógico denominado middleware que controla las conversaciones. Su función es independizar ambos procesos (cliente y servidor) La interfaz que presenta es la estándar de los servicios de red que hace que los procesos "piensen" en todo momento que se están comunicando con una red.

4.1.2 Características del modelo. En el modelo CLIENTE/SERVIDOR se puede encontrar las siguientes características:

- El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.
- Un servidor da servicio a múltiples clientes en forma concurrente.
- Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.
- La interrelación entre el hardware y el software están basados en una infraestructura poderosa, de tal forma que el acceso a los recursos de la red no muestra la complejidad de los diferentes tipos de formatos de datos y de los protocolos.
- Un sistema de servidores realiza múltiples funciones al mismo tiempo que presenta una imagen de un solo sistema a las estaciones Clientes. Esto se logra

combinando los recursos de cómputo que se encuentran físicamente separados en un solo sistema lógico, proporcionando de esta manera el servicio más efectivo para el usuario final.

- También es importante hacer notar que las funciones Cliente/Servidor pueden ser dinámicas. Ejemplo, un servidor puede convertirse en cliente cuando realiza la solicitud de servicios a otras plataformas dentro de la red.
- Su capacidad para permitir integrar los equipos ya existentes en una organización, dentro de una arquitectura informática descentralizada y heterogénea.
- Además se constituye como el nexo de unión mas adecuado para reconciliar los sistemas de información basados en mainframes o minicomputadores, con aquellos otros sustentados en entornos informáticos pequeños y estaciones de trabajo.
- Designa un modelo de construcción de sistemas informáticos de carácter distribuido.
- Su representación típica es un centro de trabajo (PC), en donde el usuario dispone de sus propias aplicaciones de oficina y sus propias bases de datos, sin dependencia directa del sistema central de información de la organización, al tiempo que puede acceder a los recursos de este host central y otros sistemas de la organización ponen a su servicio.

4.1.3 Ventajas del Modelo. Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor, es la existencia de plataformas de hardware cada vez más baratas. Esta constituye a su vez una de las más palpables ventajas de este esquema, la posibilidad de utilizar máquinas considerablemente más baratas que las requeridas por una solución centralizada, basada en sistemas grandes. Además, se pueden utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo cual contribuye considerablemente a la reducción de costos y favorece la flexibilidad en la implantación y actualización de soluciones.

El esquema Cliente/Servidor facilita la integración entre sistemas diferentes y comparte información permitiendo, por ejemplo que las máquinas ya existentes puedan ser utilizadas pero utilizando interfaces más amigables al usuario. De esta manera, se puede

integrar PCs con sistemas medianos y grandes, sin necesidad de que todos tengan que utilizar el mismo sistema operacional.

Al favorecer el uso de interfaces gráficas interactivas, los sistemas construidos bajo este esquema tienen mayor interacción más intuitiva con el usuario. El uso de interfaces gráficas para el usuario, el esquema Cliente/Servidor presenta la ventaja, con respecto a uno centralizado, de que no es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.

Una ventaja adicional del uso del esquema Cliente/Servidor es que es más rápido el mantenimiento y el desarrollo de aplicaciones, pues se pueden emplear las herramientas existentes (por ejemplo los servidores de SQL o las herramientas de más bajo nivel como los sockets o el RPC). La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones. El esquema Cliente/Servidor contribuye además, a proporcionar, a los diferentes departamentos de una organización, soluciones locales, pero permitiendo la integración de la información relevante a nivel global.

4.1.4 Aplicación en JDBC. JDBC es una API de JAVA para permitir ejecutar instrucciones SQL (Structured Query Language: Lenguaje estructurado de consultas), que es un lenguaje de alto nivel para crear, manipular, examinar y gestionar bases de datos relacionales. Para que una aplicación pueda hacer operaciones en una BD (base de datos), ha de tener el correspondiente Driver que conecte la aplicación con esta. Así pues la API JDBC es básicamente un paquete de JAVA (java.sql) que contiene un conjunto de clases e interfaces escritas en JAVA.

Ahora se puede resumir en tres frases lo que hace JDBC:

- Establece una conexión con una BD, que puede ser remota o no.
- Envía sentencias SQL a la BD.
- Procesa los resultados obtenidos de la BD.

La API JDBC soporta dos modelos distintos de acceso a las BD:

Para esto se verán los llamados Modelo de dos capas y Modelo de tres capas.

- ***Modelo de dos capas:***

  En este modelo la aplicación JAVA o el Applet, se conectan directamente con la BD. Esto significa que el driver JDBC específico para conectarse con la BD estará instalado en el sistema local. La BD puede estar en otra maquina y se accede a ella mediante red. Esta configuración también se llama Cliente/Servidor. El programa cliente envía instrucciones SQL a la BD, y esta las procesa y envía los resultados de vuelta al usuario.

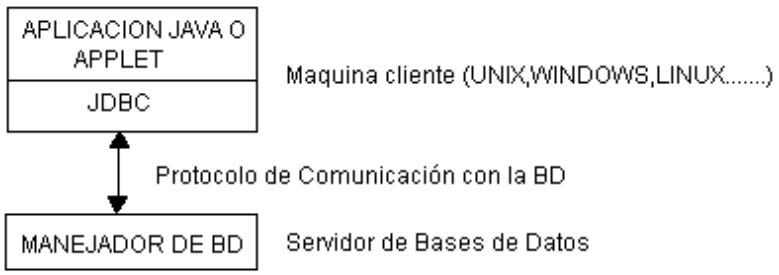

Figura 3. Modelo JDBC de dos capas

- ***Modelo de tres capas***

  En este modelo, las instrucciones son enviadas a una capa intermedia que se encarga de enviar las sentencias SQL a la Base de datos (BD). El manejador de BD procesa las sentencias y retorna los resultados a la capa intermedia que se encarga de enviarlos al usuario.

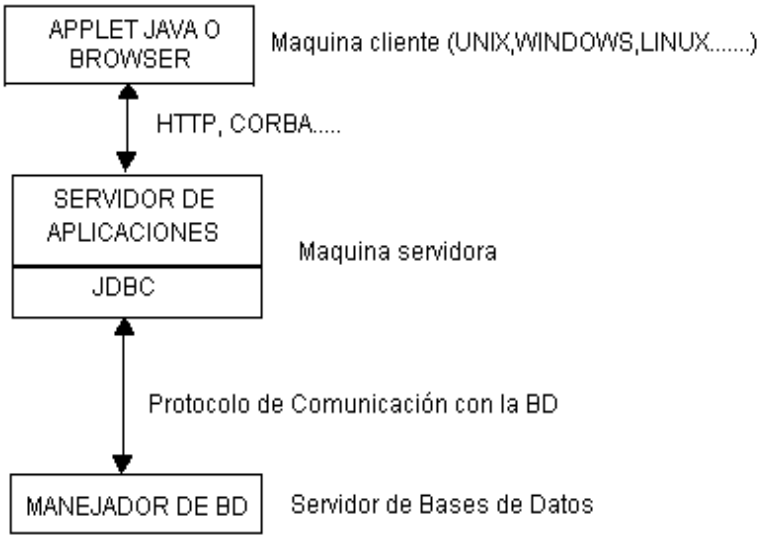

Figura 4. Modelo JDBC de tres capas

Este modelo ofrece diferentes ventajas:

a. El nivel intermedio mantiene el control del tipo de operaciones que se puede hacer en la BD.
b. Los drivers JDBC para conectarse en la BD, no han de residir en la maquina cliente.

4.2 El PATRÓN DE DISEÑO MVC (MODEL – VIEW – CONTROLLER)

Este patrón se usa en aplicaciones interactivas que requieren una interfaz de usuario flexible.

Este patrón es adecuado para situaciones donde:

- Es necesario presentar los mismos datos de diferentes formas (gráficos circulares, gráficos de barras, etc.) o utilizando diferentes interfaces de usuario (Motif, Windows 95, etc.).
- Si cambia un dato, los datos se deben de actualizar en todas las representaciones.
- El interfaz de usuario debe de ser fácil de modificar.

El patrón Modelo-Vista-Controlador (MVC) descompone una aplicación interactiva en tres grandes bloques:

El *modelo* contiene los datos y la funcionalidad de la aplicación. Es independiente de la representación de los datos.

Las *vistas* muestran la información al usuario de una cierta forma. Existen todas las que se necesite definir.

Cada *vista* tiene un *controlador* asociado. Los controladores reciben entradas en forma de eventos que responden a mandos realizados por el usuario a través del ratón o del teclado. El control traduce estos eventos a peticiones a la *vista* o al *modelo*.

4.2.1 Estructura

Clase Modelo	Colaboradores
Responsabilidad • Contiene la funcionalidad de la aplicación. • Lleva un registro de las vistas y controladores del sistema. • Notifica los cambios en los datos a los componentes.	• Vista • Controlador
Clase Controlador	Colaboradores
Responsabilidad • Acepta los eventos de entrada. • Traduce los eventos de entrada a peticiones al modelo o a las vistas. • Implementa el procedimiento actualizar si es necesario..	• Vista • Controlador
Clase Vista	Colaboradores
Responsabilidad • Crea e inicializa su controlador asociado • Muestra información al usuario. • Actualiza la información. • Recoge datos del modelo.	• Modelo • Controlador

Tabla 7. Modelo MVC

En el diseño e implementación de interfaces gráficas de usuario (GUI), MVC ha nacido como una arquitectura muy popular para particionar la funcionalidad del software.

El patrón de diseño MVC divide la funcionalidad de la aplicación en tres componentes que interactúan entre si: El Modelo, La Vista y El Controlador. A continuación se presenta el esquema de interacción de estos tres componentes:

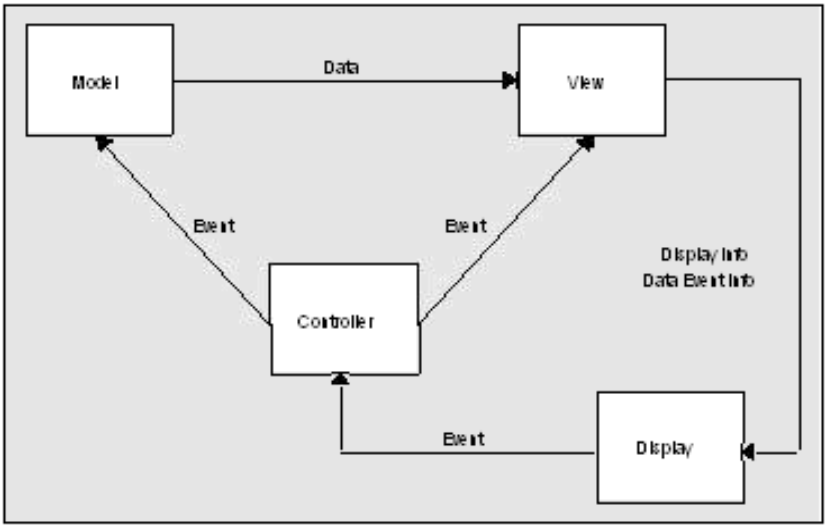

Figura 5. Interacción de los componentes: El Modelo, La Vista y El Controlador.

4.2.2 El Modelo. El modelo representa la lógica de negocios de una aplicación. El encapsulamiento de la lógica de negocio en componentes facilita las pruebas, mejora la calidad y promueve el rehusó de componentes. Es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones.[12]

El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo

El modelo puede subdividirse en Componentes de Estado y Componentes de Acción.

Los componentes de estado definen el conjunto de valores o propiedades actuales del modelo e incluye métodos para manipular dichas propiedades.

[12] BROWN, Simon. BURDICK, Robert. Professional JSP. Wrox Press. 2001. p. 100

Los componentes de acción definen los cambios permitidos al estado del modelo en respuesta a eventos. La lógica de negocios de la aplicación siempre insta a la creación de componentes de acción.

4.2.3 La Vista. La vista representa la lógica de la presentación de la aplicación. Los componentes de vista obtienen el estado actual del sistema a partir del Modelo y provee la interfaz de usuario adecuada para el protocolo en uso. Es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo.

Como parte de la generación de la interfaz gráfica de usuario, la Vista es responsable de presentar el conjunto específico de eventos que el usuario puede acceder en un momento determinado.

4.2.4 El Controlador. Es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Es el responsable de recibir los eventos, determinar el manejador de evento adecuado, invocar dicho manejador y finalmente disparar la generación de la respuesta adecuada e interactúa con el Modelo a través de una referencia al propio Modelo

El controlador debe implementar los siguientes conceptos:

- Seguridad: Debe ejecutar tareas relacionadas con la seguridad del sistema como Autenticación y autorización.
- Identificación de eventos: Debe identificar el evento que será ejecutado.
- Preparar el modelo: Debe asegurar la disponibilidad de los componentes del modelo requeridos.
- Procesar eventos: Debe asignar a cada petición el manejador de eventos adecuado e invocarlo.

- Manejo de errores: Debe encargarse de manejar cualquier error generado por los manejadores de eventos. Luego el controlador podrá redirigir el error a alguna interfaz para su presentación al usuario.
- Disparar una respuesta: Debe dirigir el control al generador de respuestas, que no es más que un elemento de la Vista para mostrar los resultados de las operaciones realizadas.

4.2.5 Ventajas

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

Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Si uno de los Componentes, posteriormente, se observa que funciona mal, puede reemplazarse sin que las otras piezas se vean afectadas. Este escenario contrasta con la aproximación monolítica típica de muchos programas Java. Todos tienen un *Frame* que contiene todos los elementos, un controlador de eventos, un montón de cálculos y la presentación del resultado. Ante esta perspectiva, hacer un cambio aquí no es nada trivial.

En este caso, la pieza central de la escena en tres dimensiones es el Modelo. El Modelo es una descripción matemática de los vértices y las caras que componen la escena. Los datos que describen cada vértice o cara pueden modificarse (quizás como resultado de una acción del usuario, o una distorsión de la escena, o un algoritmo de sombreado). Sin embargo, no tiene noción del punto de vista, método de presentación, perspectiva o fuente de luz. El Modelo es una representación pura de los elementos que componen la escena.

La porción del programa que transforma los datos dentro del Modelo en una presentación gráfica es la Vista. La Vista incorpora la visión del Modelo a la escena; es la representación gráfica de la escena desde un punto de vista determinado, bajo condiciones de iluminación determinadas.

El Controlador sabe que puede hacer el Modelo e implementa el interfase de usuario que permite iniciar la acción. En este ejemplo, un panel de datos de entrada es lo único que se necesita, para permitir añadir, modificar o borrar vértices o caras de la figura.

5. TECNOLOGIAS PARA PROGRAMACION EN WEB

Los primeros sitios Web solo eran colecciones de páginas Web enlazadas entre ellas por el Lenguaje HTML. Hoy en día, los sitios Web incluyen multimedia, aplicaciones de comercio electrónico, y otro tipo de sofisticadas aplicaciones basadas en Web como voz IP y banca online. Además de los avances en contenido, se han producido avances en las tecnologías de servidores Web, como el desarrollo de servidores de aplicaciones, y tecnologías de creación dinámica de contenido, como las páginas JSP y las páginas ASP.

5.1 JAVA

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo CPU utilizada. Desarrollan un código “neutro” que no depende del tipo de electrodoméstico, el cual se ejecuta sobre una “máquina hipotética o virtual” denominada Java Virtual Machine (JVM). La JVM interpreta el código neutro convirtiéndolo a código particular de la CPU utilizada.

Java, como lenguaje de programación para computadores, se introdujo a finales de 1995. La clave fue la incorporación de un intérprete Java en el programa Netscape Navigator, versión 2.0, produciendo una verdadera revolución en Internet. Java 1.1 apareció a principios de 1997, mejorando la primera versión del lenguaje.

Java no es solo un lenguaje de programación poderoso y construido para ser seguro, multiplataforma e internacional, si no que siempre está en constante actualización, incluyéndole nuevas características y bibliotecas de clases que manejan nuevos problemas de manera elegante y eficiente como por ejemplo multihilos, acceso a bases de datos, programación para redes y computación distribuida.[13]

5.1.1 Definición. Java es un lenguaje originalmente desarrollado por un grupo de ingenieros de Sun, Utilizado por Netscape posteriormente como base para Javascript.

Java es un lenguaje de objetos e independiente de la plataforma. Su uso se destaca en el Web y sirve para crear todo tipo de aplicaciones (locales, Intranet o Internet). El principal objetivo de Java es llegar a ser el "nexo universal" que conecte a los usuarios con la información, este está situado en el ordenador local, en un servidor de Web, en una base de datos o en cualquier otro lugar.
Es un lenguaje simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico.

Java requiere un sistema operativo multithreading (como Unix, Windows95, OS/2...)

Java es un lenguaje orientado a objetos puro, es decir, solamente se puede programar utilizando la metodología de la POO, a diferencia de otros lenguajes como C++, que además de poseer el soporte para la POO, también es posible programar en él del modo tradicional (programación estructurada).

Como cualquier lenguaje humano, Java provee los medios para expresar conceptos, y éstos se traducen en objetos a final de cuentas.

No se puede mirar a Java solo como una colección de características (algunas de ellas no tienen sentido por separado). Se puede usar la suma de las partes solamente si se piensa en *diseño*, y no solo en *codificación*.

[13] ECKEL, Bruce. Thinking in Java. 3d Edition. Prentice Hall. 2002. p. 25

Java es un lenguaje que nació de C++; tiene muchas de sus características en cuanto a sintaxis, pero la metodología y la manera interna de trabajar es muy diferente.

Java genera código multiplataforma, ya que dicho código es un código intermedio que necesita de un intérprete para ejecutarse. Esto, a simple vista es una desventaja frente a los lenguajes compilados como C++ u Object Pascal, pero Java ha llegado a ser tan popular y poderoso, que existen máquinas virtuales de Java (el intérprete del código intermedio) para casi cualquier sistema operativo imaginable actualmente. Esto, además de la flexibilidad y la potencia que ofrece (puede hacer aplicaciones independientes, aplicaciones para la Web, y muchos otros tipos de aplicaciones) hacen de Java la mejor opción a la hora de escoger un lenguaje para las aplicaciones.

Java es un producto y marca registrada de SUN Microsystems, y esta compañía ha volcado todos sus esfuerzos para estandarizar esta plataforma, generando diferentes APIs (Application Programming Interfaces), basadas en estándares, que ayudan a programar distintos tipos de aplicaciones y aplicarlas a solucionar distintos tipos de problemas, desde las aplicaciones de escritorio normales, pasando por aplicaciones Web, hasta aplicaciones distribuidas.

Una de las herramientas más poderosas de Java es su API para acceso a bases de datos relacionales. Esta API es el llamado JDBC (Java DataBase Connectivity). Además de ella, existe el estándar J2EE para aplicaciones distribuidas, que junto con JDBC conforman los pilares de este modelo de programación.

5.1.2 Características

- Robusto: Java no permite el manejo directo del hardware ni de la memoria (inclusive, no permite modificar valores de punteros, por ejemplo); el interprete siempre tiene el control.

- Gestiona la memoria automáticamente: El compilador es suficientemente inteligente como para no permitir un montón de cosas que podrían traer

problemas, como usar variables sin inicializarlas, modificar valores de punteros directamente, acceder a métodos o variables en forma incorrecta, utilizar herencia múltiple, etc.

- No permite el uso de técnicas de programación inadecuadas

- Cliente/Servidor: Está diseñado específicamente para trabajar sobre una red, de modo que incorpora objetos que permiten acceder a archivos en forma remota (vía URL por ejemplo).

- Herramientas de documentación incorporadas: Con el JDK (Java Development Kit) vienen incorporadas muchas herramientas, entre ellas un generador automático de documentación que, con un poco de atención al poner los comentarios en las clases, crea inclusive toda la documentación de las mismas en formato HTML.

- Simplicidad:Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos.
  Además elimina muchas de las características de otros lenguajes como C++, específicamente presenta características muy útiles como el garbage collector (reciclador de memoria dinámica). No es necesario preocuparse de liberar memoria, el reciclador se encarga de ello y como es un hilo de baja prioridad, cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que reduce la fragmentación de la memoria.

- Orientado a objetos: Java es un lenguaje específicamente creado para trabajar bajo la filosofía POO (Programación Orientada a Objetos), dado a que todo lo que hay en Java son objetos.
  Java soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.

- Distribuido: Java como tal no es distribuido, pero si proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas interactuando.

- Arquitectura neutral: El indicar que es de arquitectura neutral no es otra cosa que afirmar que Java puede correr sobre cualquier tipo de hardware, es decir, que es independiente de la plataforma donde se vaya a ejecutar.
  Porque en realidad Java es un lenguaje interpretado, y al compilar un programa en Java, lo que se genera es un seudo código, para una maquina virtual, la cual se conoce como JVM.

- Seguridad: Las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus. Java no posee una semántica específica para modificar la pila de programa, la memoria libre o utilizar objetos y métodos de un programa sin los privilegios del kernel del sistema operativo. Además, para evitar modificaciones por parte de los crackers de la red, implementa un método ultraseguro de autenticación por clave pública.

  Además Java implementa mecanismos de seguridad que limitan el acceso a recursos de la maquina donde se ejecuta, especialmente en el caso de los applets, esta ultima se refiere a aplicaciones que se cargan desde un servidor y se ejecutan en el cliente.

- Portable: Java facilita la portabilidad de las aplicaciones entre diferentes tipos de hardware con el mínimo esfuerzo. Por ejemplo los enteros son siempre *enteros* y además, enteros de 32 bits en complemento a 2. Además, Java construye sus interfaces de usuario a través de un sistema abstracto mas conocido como AWTT de forma tal que las ventanas, puedan ser implantadas en cualquier entorno.

- Multihilos: Java permite muchas actividades simultáneas en un programa. Los threads (a veces llamados, procesos ligeros), son básicamente pequeños procesos o piezas independientes de un gran proceso. La ventaja de ser Multihilo consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de

flujo único de programa (Mono-Hilo) tanto en facilidad de desarrollo como en rendimiento.

5.1.3 Ventajas

- Es un lenguaje orientado a objetos.
- Los objetos cumplen con las características de encapsulación y herencia.
- No hay programas que actúen modificando al objeto, éste se mantiene en cierto modo independiente del resto de la aplicación.
- Proporciona varios objetos y métodos para obtener información sobre los objetos dentro de la base de datos.
- Alto grado de adaptabilidad, extensibilidad y genericidad.
- Incorpora muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.).
- Soporte completo para bases de datos por medio de JDBC.

5.1.4 Desventajas

- Por ser un lenguaje interpretado, la ejecución de los programas es un poco más lenta que los compilados.
- El código objeto generado requiere que el sistema operativo donde se quiera ejecutar tenga instalada la Máquina Virtual de Java para dicho sistema.
- Los programas en Java requieren bastante memoria para ejecutarse, dado el hecho que necesita reservar memoria adicional para la MVJ.

5.2 JAVA SCRIPTS

JavaScript es el lenguaje que permite interactuar con el navegador de manera dinámica y eficaz, proporcionando a las páginas Web dinamismo y vida.

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

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

5.2.1 Versiones

El programa que va a interpretar los programas JavaScript es el propio navegador, lo que significa que si el usado no soporta JavaScript, no se podrá ejecutar las funciones que se programen. Desde luego, Netscape y Explorer lo soportan, el primero desde la versión 2 y el segundo desde la versión 3.

5.3 JAVA SERVER PAGE (JSP)

JSP es una tecnología basada en Java que simplifica el proceso de desarrollo de sitios web dinámicos. Las *Java server Pages* son ficheros de texto que sustituyen a las páginas HTML tradicionales. Los ficheros JSP contienen etiquetas HTML y código embebido que permite al diseñador de la página web acceder a datos desde código Java que se ejecuta en el servidor.

JSP se implementa utilizando la tecnología *Servlet*. Cuando un servidor web recibe una petición de una página .jsp, la redirecciona a un proceso especial dedicado a manejar la ejecución de *servlets* (*servlet container*) llamado JSP container.

JSP es una tecnología simple pero poderosa, usada para generar HTML de forma dinámica, del lado del servidor. Son una extensión directa de los Servlets de Java y dan una manera de separar la generación de contenido, de la presentación del mismo.[14]

[14] GOODWILL, James. Pure JSP, Java Server Pages. SAMS Publishing. 2001. p. 8

JSP es muy similar a ASP o PHP en su manera de programarse, ya que se mezcla con el código HTML mediante *tags* o *marcas* que indican que se trata de código Java, y así lo entiende el servidor de aplicaciones sobre el cual se distribuyen.

Las páginas escritas usando JSP, llevan la extensión .jsp para distinguirlas de las páginas estáticas normales. La sintaxis JSP cumple con el estándar XML de formación de etiquetas.

JavaServer Pages, como se dijo anteriormente, son una extensión de los Servlets de Java, por lo tanto hay que entender bien los Servlets para poder comprender el funcionamiento de los JSP. Los Servlets son extensiones genéricas de los servidores basados en Java. Su uso más común es extender la funcionalidad de los servidores Web, proveyendo un reemplazo eficiente, portable y fácil de usar para los ya bastante conocidos CGI (Common Gateway Interface). Un Servlet es un módulo de carga dinámica que esperan peticiones desde el servidor Web. Dado que los Servlets corren del lado del servidor, no hay problemas de compatibilidad con los navegadores Web

5.3.1 Beneficios de JSP

- Mejoras en el rendimiento: Utilización de hilos Java para el manejo de las peticiones. El contenedor *servlet* puede ser ejecutado como parte del servidor web. Como se puede tener hilos con el mismo padre se podrá compartir recursos con facilidad entre las peticiones.
- Soporte de componentes reutilizables: Creación y utilización de *JavaBeans* del servidor. Los *JavaBeans* utilizados en páginas .jsp pueden ser utilizados en *servlets, applets* o aplicaciones Java.
- Separación entre la programación y la presentación: Los cambios realizados en el diseño de las páginas no interfieren en la lógica de la programación y viceversa.

Para poder utilizar esta tecnología es necesario un servidor web que soporte a páginas html y código que implemente un contenedor JSP donde ejecutar las etiquetas JSP. Existen servidores web que incorporan dicha capacidad dentro de su código (Netscape Enterprise y Aplication Server 4.0) así como servidores escritos íntegramente en Java

(Java Web Server de Sun y Jigsaw de W3 Consortium) que dan soporte directamente a esta tecnología.

5.4 SERVLETS

Los Servlets son módulos que extienden los servidores orientados a petición - respuesta, como los servidores web compatibles con Java. Por ejemplo, un servlet podría ser responsable de tomar los datos de un formulario de entrada de pedidos en HTML y aplicarle la lógica de negocios utilizada para actualizar la base de datos de pedidos de la compañía.

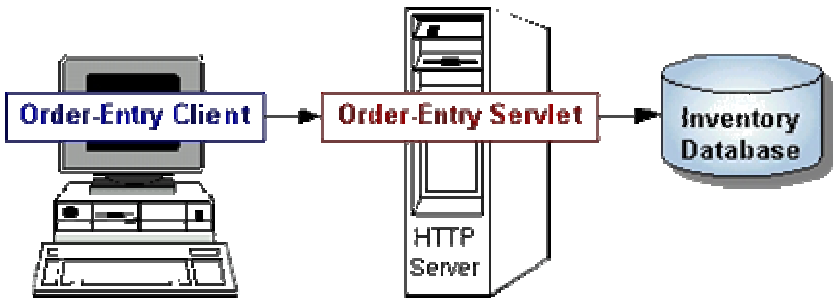

Figura 6. Servlets

Los Servlets son para los servidores lo que los applets son para los navegadores. Sin embargo, al contrario que los applets, los servlets no tienen interfase gráfico de usuario.

Los servelts pueden ser incluidos en muchos servidores diferentes porque el API Servlet, el que se utiliza para escribir Servlets, no asume nada sobre el entorno o protocolo del servidor. Los servlets se están utilizando ampliamente dentro de servidores HTTP; muchos servidores Web soportan el API Servlet.

Los servlets son una alternativa más eficiente, segura, poderosa, portable y fácil de usar que los CGI tradicionales y similares. [15]

[15] HALL, Marty. Core Servlets and JavaServer Pages. Prentice Hall and SUN Microsystems. 2003. p. 7

5.4.1 Configurar Servlets. Los Servlets se registran o configuran como una parte de una Aplicación Web. Para registrar un servlet, se añadieron varias entradas al descriptor de despliegue de la Aplicación Web. La primera entrada bajo, el elemento <servlet> define el nombre del servlet y la clase compilada que ejecuta el servlet. Este elemento también contiene definiciones para los parámetros de inicialización y roles de seguridad del servlet. La segunda entrada, bajo el elemento <servlet-mapping> define el patrón URL que llama este servlet.

5.4.2 El Ciclo de Vida de un Servlet. Cada servlet tiene el mismo ciclo de vida.

- Un servidor carga e inicializa el servlet.
- El servlet maneja cero o más peticiones de cliente.
- El servidor elimina el servlet. (Algunos servidores sólo cumplen este paso cuando se desconectan). [16]

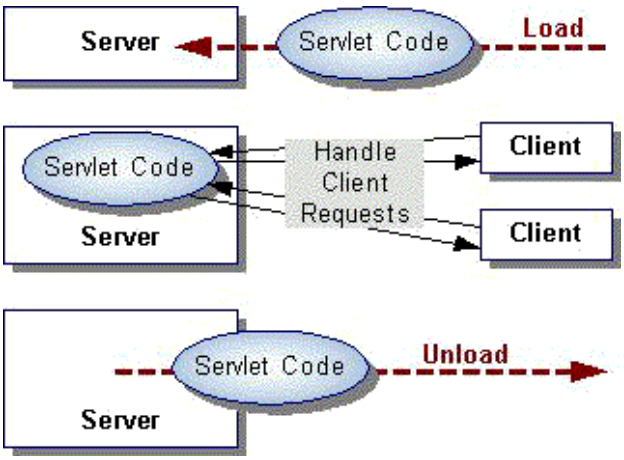

Figura 7. Ciclo de vida de un servlet

- **Inicializar un Servlet**

  Cuando un servidor carga un servlet, ejecuta el método **init** del servlet. La inicialización se completa antes de manejar peticiones de clientes y antes de que el servlet sea destruido.

  Aunque muchos servlets se ejecutan en servidores multi-hilos, los servlets no tienen problemas de concurrencia durante su inicialización. El servidor llama sólo

[16] HUNTER, Jason. Java Servlets Programming. O'Reilly. 1998. p. 48

una vez al método **init**, cuando carga el servlet, y no lo llamará de nuevo a menos que vuelva a recargar el servlet. El servidor no puede recargar un servlet sin primero haber destruido el servlet llamando al método **destroy**.

- **Interactuar con Clientes**
  Después de la inicialización, el servlet puede manejar peticiones de clientes.

- **Destruir un Servlet**
  Los servlets se ejecutan hasta que el servidor los destruye, por ejemplo, a petición del administrador del sistema. Cuando un servidor destruye un servlet, ejecuta el método **destroy** del propio servlet. Este método sólo se ejecuta una vez. El servidor no ejecutará de nuevo el servlet, hasta haberlo cargado e inicializado de nuevo.
  Mientras se ejecuta el método destroy, otro hilo puede estar ejecutando una petición de servicio.

5.5 JAVA BEANS

5.5.1 Definición de Java Bean. Son componentes de software escritos en Java. Los componentes en si son llamados Beans y se adhieren a las especificaciones de la API de JavaBeans.[17]

Aunque los beans pueden variar ampliamente en funcionalidad desde los más simples a los más complejos, todos ellos comparten las siguientes características:

- **Introspection**: Permite analizar a la herramienta de programación o IDE como trabaja el bean

- **Customization**: El programador puede alterar la apariencia y la conducta del bean.

- **Events**: Informa al IDE de los sucesos que puede generar en respuesta a las acciones del usuario o del sistema, y también los sucesos que puede manejar.

[17] FIELDS, Duane K. KOLB, Mark. Web Development with JavaServer Pages. Manning. 2000. p. 117

- **Properties**: Permite cambiar los valores de las propiedades del bean para personalizarlo (customization).

- **Persistence**: Se puede guardar el estado de los beans que han sido personalizados por el programador, cambiando los valores de sus propiedades.

En general, un bean es una clase que obedece ciertas reglas:

- Un bean tiene que tener un constructor por defecto (sin argumentos)

- Un bean tiene que tener persistencia, es decir, implementar el interface *Serializable*.

- Un bean tiene que tener introspección (**instrospection**). Los IDE reconocen ciertas pautas de diseño, nombres de las funciones miembros o métodos y definiciones de las clases, que permiten a la herramienta de programación mirar dentro del bean y conocer sus propiedades y su conducta.

5.5.2 Propiedades. Una propiedad es un atributo del JavaBean que afecta a su apariencia o a su conducta. Por ejemplo, un botón puede tener las siguientes propiedades: el tamaño, la posición, el título, el color de fondo, el color del texto, si está o no habilitado, etc.

Las propiedades de un bean pueden examinarse y modificarse mediante métodos o funciones miembro, que acceden a dicha propiedad, y pueden ser de dos tipos:

- **get**ter method: lee el valor de la propiedad
- **set**ter method: cambia el valor de la propiedad.

Un IDE que cumpla con las especificaciones de los JavaBeans sabe como analizar un bean y conocer sus propiedades. Además, crea una representación visual para cada uno de los tipos de propiedades, denominada editor de propiedades, para que el programador pueda modificarlas fácilmente en el momento del diseño.

Cuando un programador, escoge un bean de la paleta de componentes y lo deposita en un panel, el IDE muestra el bean sobre el panel. Cuando se selecciona el bean aparece una hoja de propiedades, que es una lista de las propiedades del bean, con sus editores asociados para cada una de ellas. El IDE llama a los métodos o funciones miembro que empiezan por **get**, para mostrar en los editores los valores de las propiedades. Si el programador cambia el valor de una propiedad se llama a un método cuyo nombre empieza por **set**, para actualizar el valor de dicha propiedad y que puede o no afectar al aspecto visual del bean en el momento del diseño.

Las especificaciones JavaBeans definen un conjunto de convenciones (design patterns) que el IDE usa para inferir qué métodos corresponden a propiedades.
public void **set**NombrePropiedad(TipoPropiedad valor)
public TipoPropiedad **get**NombrePropiedad( )

Cuando el IDE carga un bean, usa el mecanismo denominado *reflection* para examinar todos los métodos, fijándose en aquellos que empiezan por **set** y **get**. El IDE añade las propiedades que encuentra a la hoja de propiedades para que el programador personalice el bean.

5.6 JAVA DATABASE CONNECTIVITY (JDBC)

JDBC (Java Database Connectivity) es una interfaz estándar de Java para conectarse a bases de datos relacionales.[18]

Se puede decir que JDBC es para Java lo que ODBC es para Windows. Windows define el estándar ODBC consistente en un conjunto de primitivas que cualquier driver o fuente ODBC debe ser capaz de entender y manipular. Los programadores que a su vez deseen escribir programas para manejar bases de datos genéricas en Windows utilizan las llamadas ODBC. Con JDBC ocurre exactamente lo mismo: JDBC es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea. Lógicamente, al igual

[18] HANES, Elizabeth. SANKO, Mike y otros. Oracle 9i JDBC Developer's Guide and Reference. Oracle Press. 2002. p. 1-2

que ODBC, la aplicación de Java debe tener acceso a un driver JDBC adecuado. Este driver es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

La necesidad de JDBC, a pesar de la existencia de ODBC, viene dada porque ODBC es un interfaz escrito en lenguaje C, que al no ser un lenguaje portable, haría que las aplicaciones Java también perdiesen la portabilidad. Y además, ODBC tiene el inconveniente de que se ha de instalar manualmente en cada máquina; al contrario que los drivers JDBC, que al estar escritos en Java son automáticamente instalables, portables y seguros.

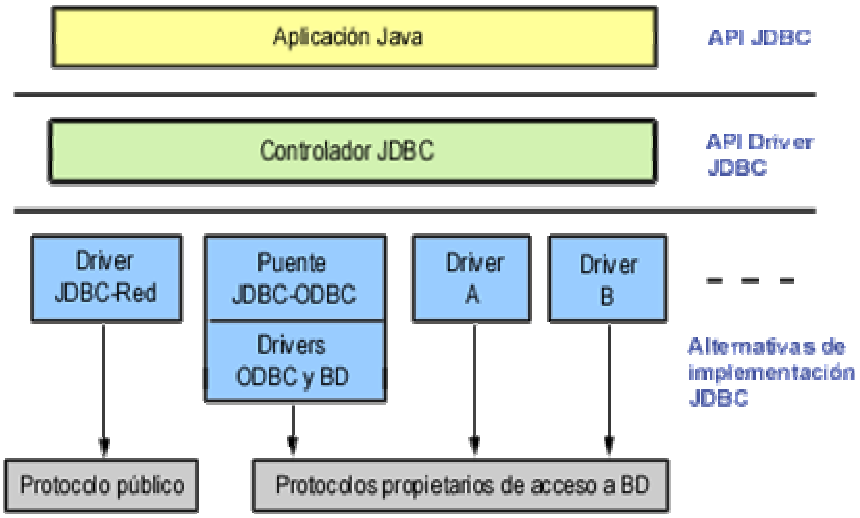

Figura 8. Conectividad con JDBC

Toda la conectividad de bases de datos de Java se basa en sentencias SQL, por lo que se hace imprescindible un conocimiento adecuado de SQL para realizar cualquier clase de operación de bases de datos. Aunque, afortunadamente, casi todos los entornos de desarrollo Java ofrecen componentes visuales que proporcionan una funcionalidad suficientemente potente sin necesidad de que sea necesario utilizar SQL, aunque para usar directamente el JDK se haga imprescindible. La especificación JDBC requiere que cualquier driver JDBC sea compatible con al menos el nivel «de entrada» de ANSI SQL 92 (ANSI SQL 92 Entry Level).

5.6.1 Acceso de JDBC a Bases de Datos

El API JDBC soporta dos modelos diferentes de acceso a Bases de Datos, los modelos de dos y tres capas.

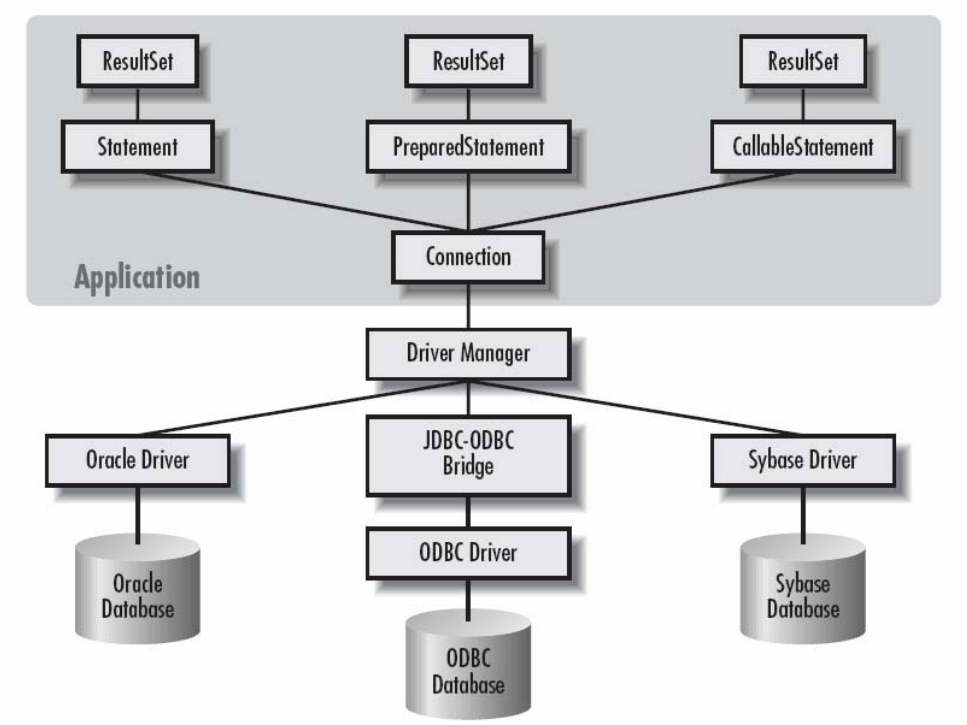

Figura 9. Modelo General.

Este modelo se basa en que la conexión entre la aplicación Java o el applet que se ejecuta en el navegador, se conectan directamente a la base de datos.

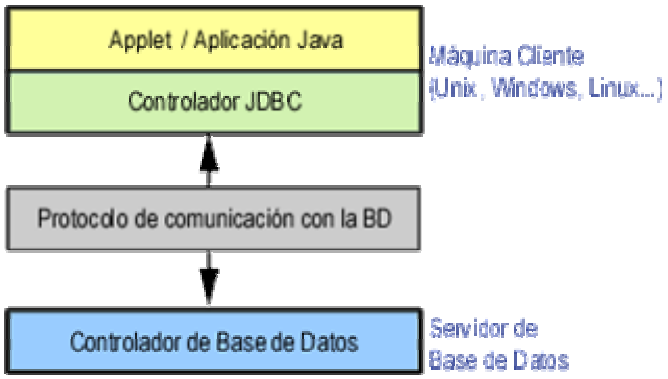

Figura 10 Modelo de dos capas.

Esto significa que el driver JDBC específico para conectarse con la base de datos, debe residir en el sistema local. La base de datos puede estar en cualquier otra máquina y se accede a ella mediante la red. Esta es la configuración de típica Cliente/Servidor: el programa cliente envía instrucciones SQL a la base de datos, ésta las procesa y envía los resultados de vuelta a la aplicación.

Modelo de tres capas

En este modelo de acceso a las bases de datos, las instrucciones son enviadas a una capa intermedia entre Cliente y Servidor, que es la que se encarga de enviar las sentencias SQL a la base de datos y recoger el resultado desde la base de datos. En este caso el usuario no tiene contacto directo, ni a través de la red, con la máquina donde reside la base de datos.

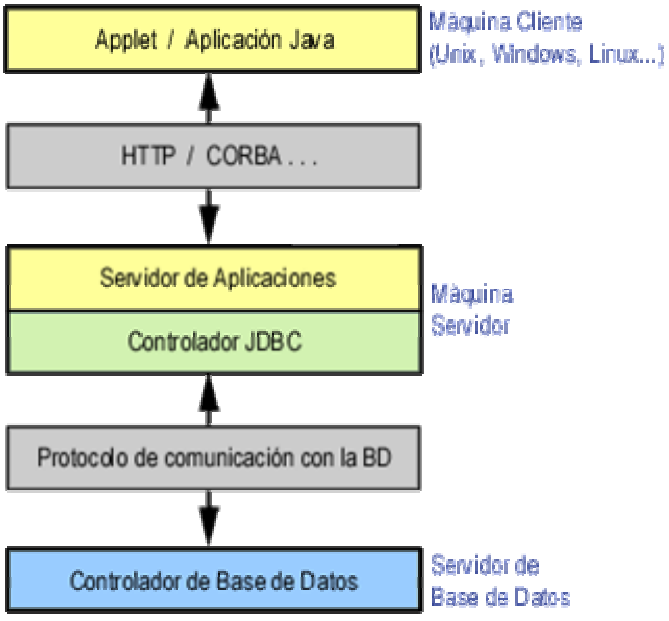

Figura 11. Modelo de tres capas.

Este modelo presenta la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan contra la base de datos, y además, está la ventaja adicional de que los drivers JDBC no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de driver.

5.6.2 Tipos De Drivers

Un driver JDBC puede pertenecer a una de cuatro categorías diferentes en cuanto a la forma de operar.

Puente JDBC-ODBC. La primera categoría de drivers es la utilizada por Sun inicialmente para popularizar JDBC y consiste en aprovechar todo lo existente, estableciendo un puente entre JDBC y ODBC. Este driver convierte todas las llamadas JDBC a llamadas ODBC y realiza la conversión correspondiente de los resultados.

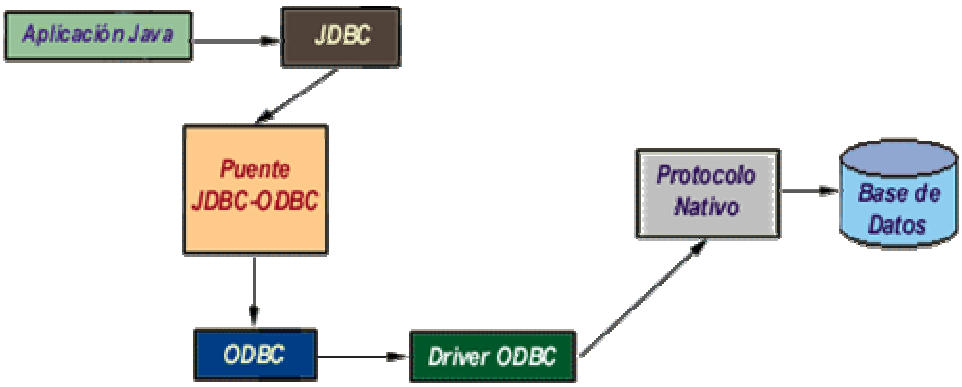

Figura 12. Puente JDBC-ODBC

La ventaja de este driver, que se proporciona con el JDK, es que Java dispone de acceso inmediato a todas las fuentes posibles de bases de datos y no hay que hacer ninguna configuración adicional aparte de la ya existente. No obstante, tiene dos desventajas muy importantes; por un lado, la mayoría de los drivers ODBC a su vez convierten sus llamadas a llamadas a una librería nativa del fabricante DBMS, con lo cual la lentitud del driver JDBC-ODBC puede ser exasperante, al llevar dos capas adicionales que no añaden funcionalidad alguna; y por otra parte, el puente JDBC-ODBC requiere una instalación ODBC ya existente y configurada.

Lo anterior implica que para distribuir con seguridad una aplicación Java que use JDBC habría que limitarse en primer lugar a entornos Windows (donde está definido ODBC) y en segundo lugar, proporcionar los drivers ODBC adecuados y configurarlos correctamente.

Esto hace que este tipo de drivers esté totalmente descartado en el caso de aplicaciones comerciales, e incluso en cualquier otro desarrollo, debe ser considerado como una solución transitoria, porque el desarrollo de drivers totalmente en Java hará innecesario el uso de estos puentes.

Java/Binario. Este driver se salta la capa ODBC y habla directamente con la librería nativa del fabricante del sistema DBMS (como pudiera ser DB-Library para Microsoft SQL Server o CT-Lib para Sybase SQL Server). Este driver es un driver 100% Java pero aún así necesita la existencia de un código binario (la librería DBMS) en la máquina del cliente, con las limitaciones y problemas que esto implica.

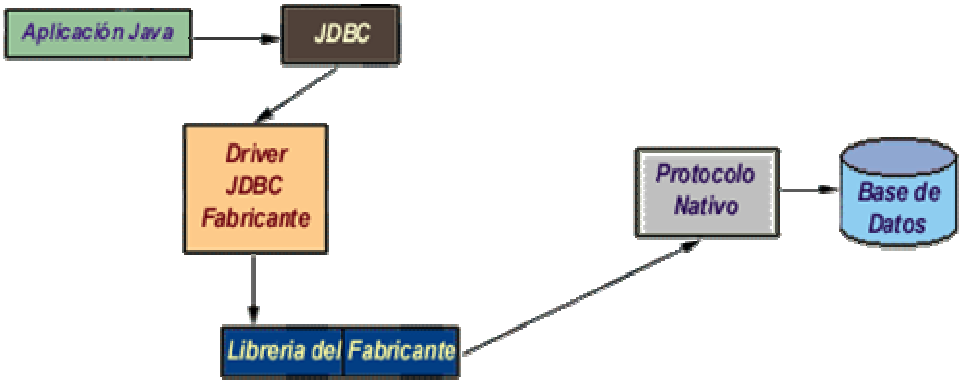

Figura 13. Modelo- *Java/Binario*

100% Java/Protocolo nativo (Usado para la aplicación). Es un driver realizado completamente en Java que se comunica con el servidor DBMS utilizando el protocolo de red nativo del servidor. De esta forma, el driver no necesita intermediarios para hablar con el servidor y convierte todas las peticiones JDBC en peticiones de red contra el servidor. La ventaja de este tipo de driver es que es una solución 100% Java y, por lo tanto, independiente de la máquina en la que se va a ejecutar el programa.

Figura 14. Modelo 100% Java/Protocolo Nativo

Igualmente, dependiendo de la forma en que esté programado el driver, puede no necesitar ninguna clase de configuración por parte del usuario. La única desventaja de este tipo de drivers es que el cliente está ligado a un servidor DBMS concreto, ya que el protocolo de red que utiliza MS SQL Server por ejemplo no tiene nada que ver con el utilizado por DB2, PostGres u Oracle. La mayoría de los fabricantes de bases de datos han incorporado a sus propios drivers JDBC del segundo o tercer tipo, con la ventaja de que no suponen un coste adicional.

100% Java/Protocolo independiente. Esta es la opción más flexible, se trata de un driver 100% Java / Protocolo independiente, que requiere la presencia de un intermediario en el servidor. En este caso, el driver JDBC hace las peticiones de datos al intermediario en un protocolo de red independiente del servidor DBMS. El intermediario a su vez, que está ubicado en el lado del servidor, convierte las peticiones JDBC en peticiones nativas del sistema DBMS. La ventaja de este método es inmediata: el programa que se ejecuta en el cliente, y aparte de las ventajas de los drivers 100% Java, también presenta la independencia respecto al sistema de bases de datos que se encuentra en el servidor.

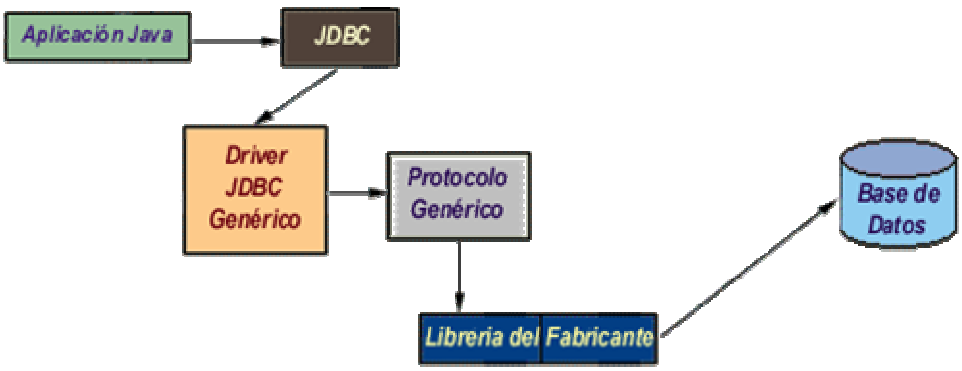

Figura 15. Modelo *100% Java/Protocolo independiente.*

De esta forma, si una empresa distribuye una aplicación Java para que sus usuarios puedan acceder a su servidor MS SQL y posteriormente decide cambiar el servidor por Oracle, PostGres o DB2, no necesita volver a distribuir la aplicación, sino que únicamente debe reconfigurar la aplicación residente en el servidor que se encarga de transformar las peticiones de red en peticiones nativas. La única desventaja de este tipo de drivers es que la aplicación intermediaria es una aplicación independiente que suele tener un coste adicional por servidor físico, que hay que añadir al coste del servidor de bases de datos.

5.7 COMBINANDO SERVLETS, JSP, Y JAVABEANS

Anteriormente se han explicado las características básicas de la arquitectura MVC. Cuando se usa este modelo se divide el software en las tres capas que lo componen (Model, View y Controller) de la siguiente manera:

- Model – Se usan Beans para la encapsulación de la lógica del software
- View – Se usa JSP para la capa de presentación o front-end
- Controller – Se usan Servlets para administrar los eventos.

Esta segregación en capas facilita el mantenimiento e incrementa la facilidad de introducir nuevas funcionalidades al sistema.

5.8 OTRAS TECNOLOGÍAS DE PROGRAMACIÓN WEB

A continuación se verá otras tecnologías de programación web diferentes a la utilizada en este proyecto:

5.8.1 ASP

ASP (Active Server Pages) es la tecnología desarrollada por Microsoft para la creación de páginas dinámicas del servidor. ASP se escribe en la misma página web, utilizando el lenguaje Visual Basic Script o Jscript (Javascript de Microsoft).
Un lenguaje del lado del servidor es aquel que se ejecuta en el servidor web, justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en

el servidor pueden realizar accesos a bases de datos, conexiones en red, y otras tareas para crear la página final que verá el cliente. El cliente solamente recibe una página con el código HTML resultante de la ejecución de la página ASP. Como la página resultante contiene únicamente código HTML, es compatible con todos los navegadores.

El tipo de servidores que emplean este lenguaje son, evidentemente, todos aquellos que funcionan con sistema Windows NT, aunque también se puede utilizar en un PC con windows 98 si se instala un servidor denominado Personal Web Server. Incluso en sistemas Linux se puede utilizar las ASP instalando un componente denominado Chilisoft.

Con las ASP es posible realizar muchos tipos de aplicaciones distintas. Permite acceso a bases de datos, al sistema de archivos del servidor y en general a todos los recursos que tenga el propio servidor. También brinda la posibilidad de comprar componentes ActiveX fabricados por distintas empresas de desarrollo de software que sirven para realizar múltiples usos, como el envío de correo, generar gráficas dinámicamente, etc.

5.8.2 PHP

PHP proviene de Hipertext Preprocesor. Es un lenguaje de programación del lado del servidor gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación.

PHP se escribe dentro del código HTML, lo que lo hace realmente fácil de utilizar, al igual que ocurre con el ASP de Microsoft, pero con algunas ventajas como su gratuidad, independencia de plataforma, rapidez y seguridad. Cualquiera puede descargar a través de la página principal de PHP www.php.net y de manera gratuita, un módulo que hace que el servidor web comprenda los scripts realizados en este lenguaje. Es independiente de plataforma, puesto que existe un módulo de PHP para casi cualquier servidor web. Esto hace que cualquier sistema pueda ser compatible con el lenguaje y significa una ventaja importante, ya que permite portar el sitio desarrollado en PHP de un sistema a otro sin prácticamente ningún trabajo.

PHP, en el caso de estar montado sobre un servidor Linux u Unix, es más rápido que ASP, dado que se ejecuta en un único espacio de memoria y esto evita las comunicaciones entre componentes COM que se realizan entre todas las tecnologías implicadas en una página ASP.

En muchas ocasiones PHP se encuentra instalado sobre servidores Unix o Linux, que son de sobra conocidos como más veloces y seguros que el sistema operativo donde se ejecuta las ASP, Windows NT o 2000. Además, PHP permite configurar el servidor de modo que se permita o rechacen diferentes usos, lo que puede hacer al lenguaje más o menos seguro dependiendo de las necesidades de cada cual.

Este lenguaje de programación está preparado para realizar muchos tipos de aplicaciones web gracias a la extensa librería de funciones con la que está dotado. La librería de funciones cubre desde cálculos matemáticos complejos hasta tratamiento de conexiones de red, por poner dos ejemplos.

Algunas de las más importantes capacidades de PHP son: compatibilidad con las bases de datos más comunes, como MySQL, Oracle, Informix, y ODBC, por ejemplo. Incluye funciones para el envío de correo electrónico, upload de archivos, crear dinámicamente en el servidor imágenes en formato GIF, incluso animadas y una lista interminable de utilidades adicionales. Su principal desventaja es que la mayoria de las veces la programación queda amarrada al motor de BD pues las funcionoes varian de acuerdo al Sistema de gestión usado.

5.8.3 PERL

Es un lenguaje de programación muy utilizado para construir aplicaciones CGI para el web. Perl proviene de Practical Extracting and Reporting Languaje, que viene a indicar que se trata de un lenguaje de programación muy práctico para extraer información de archivos de texto y generar informes a partir del contendido de los ficheros.

Es un lenguaje libre de uso, eso quiere decir que es gratuito. Antes estaba muy asociado a la plataforma Unix, pero en la actualidad está disponible en otros sistemas operativos como Windows.

Perl es un lenguaje de programación interpretado, al igual que muchos otros lenguajes de Internet como Javascript o ASP. Esto quiere decir que el código de los scripts en Perl no se compila sino que cada vez que se quiere ejecutar se lee el código y se pone en marcha interpretando lo que hay escrito. Además es extensible a partir de otros lenguajes, ya que desde Perl se podrá hacer llamadas a subprogramas escritos en otros lenguajes. También desde otros lenguajes se podrá ejecutar código Perl.

Perl está inspirado a partir de lenguajes como C, sh, awk y sed (algunos provenientes de los sistemas Uníx), pero está enfocado a ser más práctico y fácil que estos últimos. Es por ello que un programador que haya trabajado con el lenguaje C y los otros tendrá menos problemas en entenderlo y utilizarlo rápidamente. Una diferencia fundamental de Perl con respecto a los otros lenguajes es que no limita el tamaño de los datos con los que trabaja, el límite lo pone la memoria que en ese momento se encuentre disponible.

Para trabajar con Perl es necesario tener instalado el intérprete del lenguaje. A partir de ese momento se puede ejecutar CGIs en los servidores web. El proceso para conseguirlo puede variar de unos servidores a otros, pero se suelen colocar en un directorio especial del servidor llamado cgi-bin donde se han colocado los correspondientes permisos CGI. Además, los archivos con el código también deberán tener permiso de ejecución.

6. DISEÑO DE LA APLICACIÓN OBJETO DE LA TESIS

La aplicación Web MultiSQL, es una aplicación diseñada e implementada utilizando el lenguaje de programación Java, que interactúa con HTML y Javascript para la presentación de los datos en la Web.

Se utilizó el patrón de diseño MVC (Model – View - Controller) para su implementación puesto que este modelo de programación permite la modularización total de la aplicación, separando la presentación de los datos del modelo de datos, lo cual acepta realizar cambios en cualquiera de los módulos sin tener que rediseñar gran parte de la aplicación.

El flujo de la aplicación y la interacción con el usuario se lleva de la siguiente manera:

Al usuario se le despliega información en la ventana del navegador (Vista o View) y se le brindan enlaces o cualquier otro elemento de interacción para que genere los eventos necesarios para controlar la aplicación. Cuando el usuario pulsa un botón o da clic en un enlace, se genera un evento que es capturado por el Controlador (Controller) quien decide, según los parámetros que se reciben desde la Vista, el manejador de evento adecuado para cumplir con su labor (Componentes de Acción del Model). Este componente de acción o manejador de evento recibe la información necesaria para realizar su labor y puede utilizar adicionalmente componentes de estado para el almacenamiento de información de estado del modelo y poderlos pasar de regreso al usuario a través del Controlador; éste recibe la información del manejador de eventos y recibe también el componente o la página de la vista que visualizará el resultado. De esta manera se completa el ciclo quedando la aplicación a la espera de que se genere otro evento por la interacción con el usuario.

6.1 DISEÑO UML DE LA APLICACIÓN

6.1.1 Diagramas de Casos de uso

Los siguientes diagramas muestran como interactúan los agentes y las operaciones globales que se realizan sobre la aplicación.

Las tareas globales que se realizan sobre el sistema por los actores Alumno y Administrador se muestran en la figura 16. Las tareas específicas que realiza el actor Administrador, como ingresar al sistema, crear usuarios o editarlos, entre otras, se muestran en la figura 17.

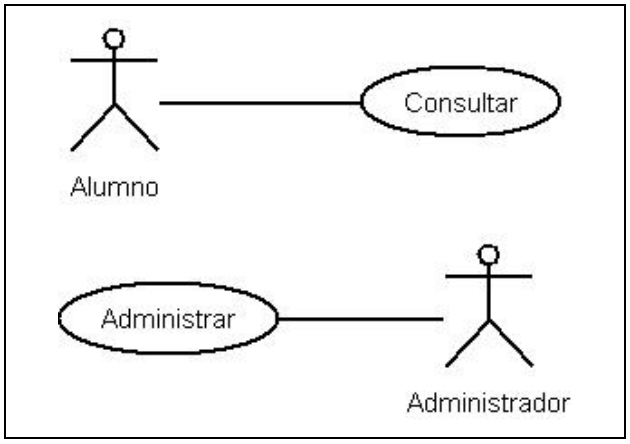

Figura 16. Diagrama general

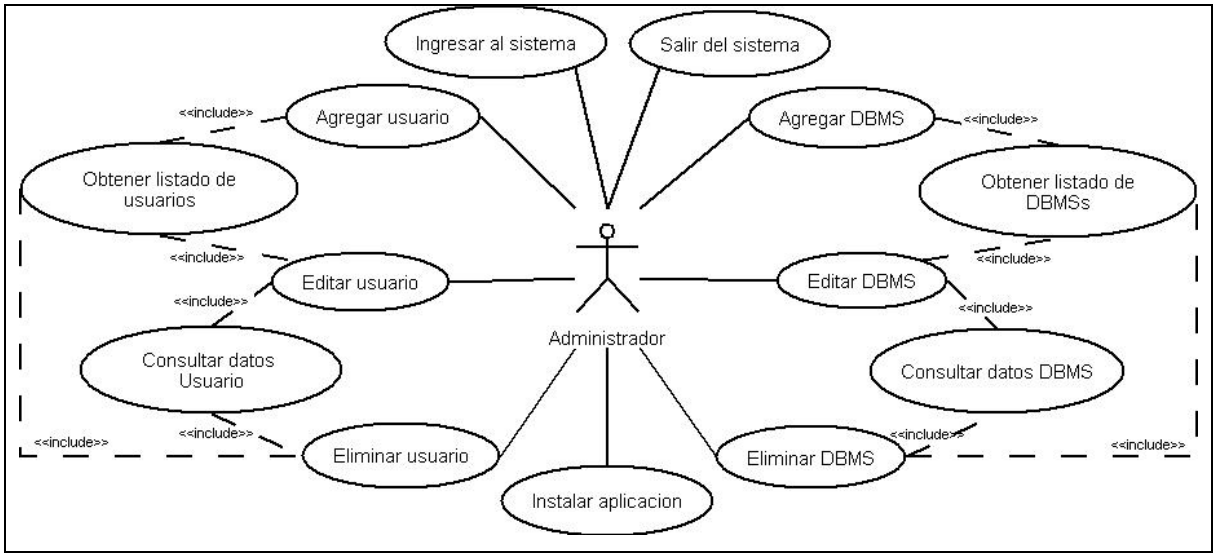

Figura 17. Diagrama del actor Administrador

El actor Alumno realiza varias tareas sobre el sistema, relacionadas con las consultas SQL. Estas tareas se muestran en la figura 18.

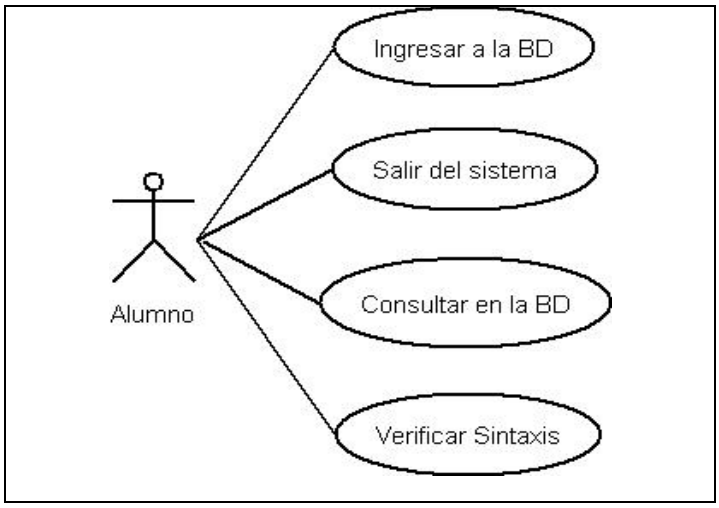

Figura 18. Diagrama del actor Alumno

6.1.2 Diagramas de Secuencia

A continuación mostraremos los diagramas de secuencia que explican más detalladamente los casos de uso para reflejar cómo interactúan los componentes para llevar a cabo cada acción.

El primer paso que realiza el administrador del sistema es instalar la aplicación misma, la cual tiene un sistema de instalación web cuyo flujo se muestra en la figura 19.

Luego mostraremos dos situaciones distintas que se pueden presentar cuando el Administrador intenta ingresar al sistema luego de realizada la instalación. Una de ellas es cuando las credenciales de ingreso son válidas y el sistema le otorga acceso exitosamente. Esta situación se observa más detalladamente en la figura 20.

Por otra parte, existe una situación en la que el Administrador no ingresa las credenciales válidas al sistema y por esta razón el sistema lo rechaza por ser un usuario no registrado. Esta situación se muestra en la figura 21.

Por último, cuando el Administrador digita bien su nombre de usuario, pero yerra en la contraseña, el sistema tampoco le da acceso al mismo y muestra el tipo de error ocurrido. Esta situación se visualiza en la figura 22.

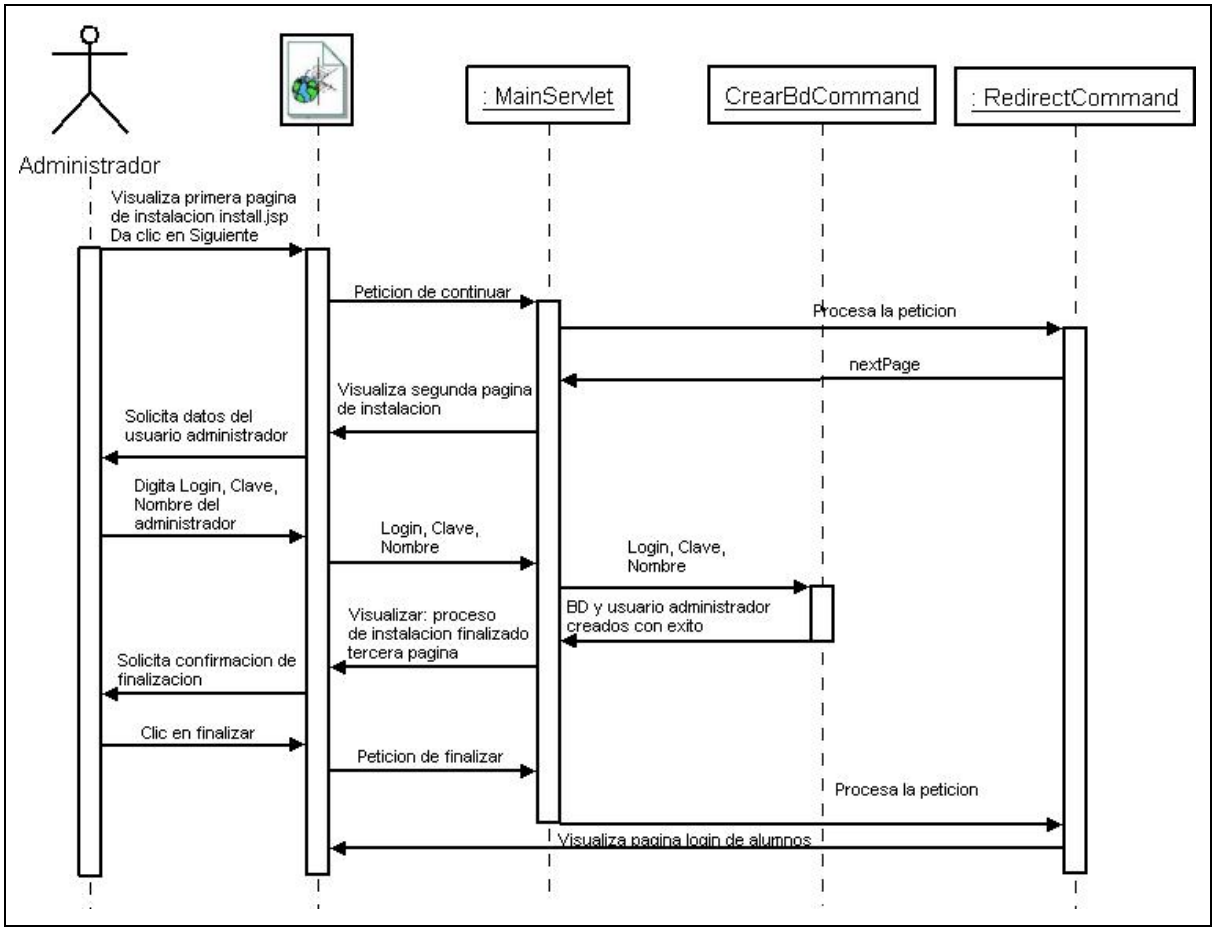

Figura 19. Diagrama del caso de uso Instalar el sistema

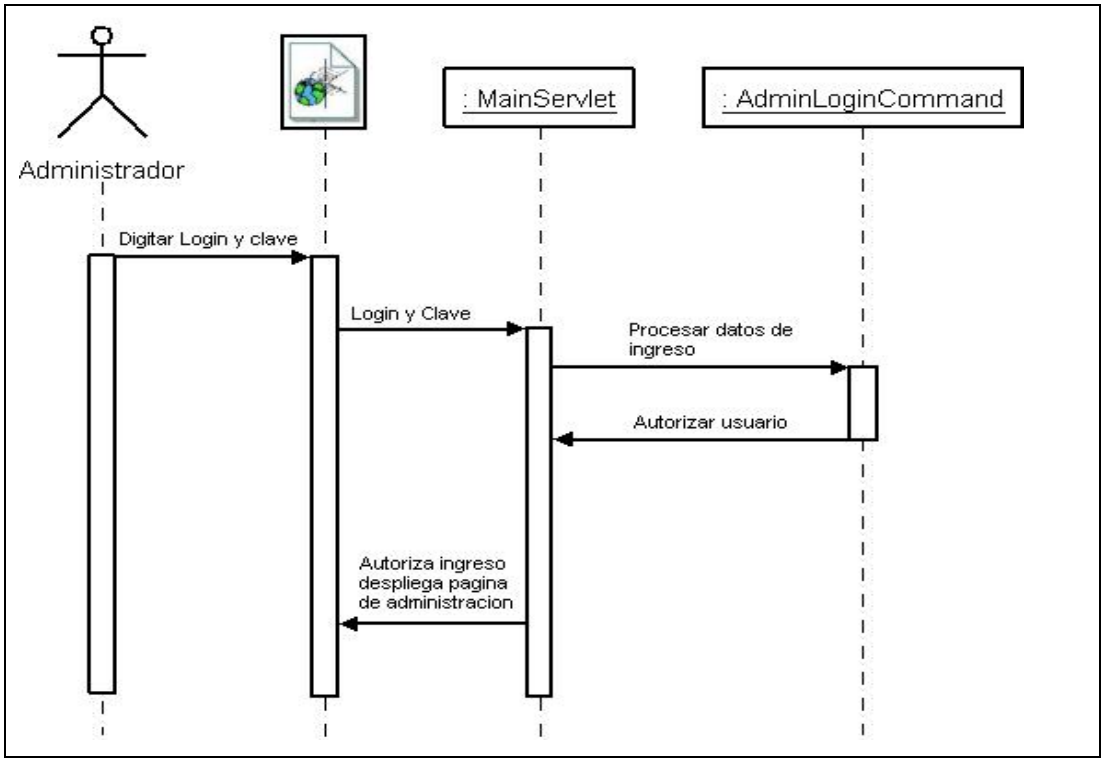

Figura 20 . Diagrama del caso de uso Ingresar al sistema Flujo Normal (Administrador)

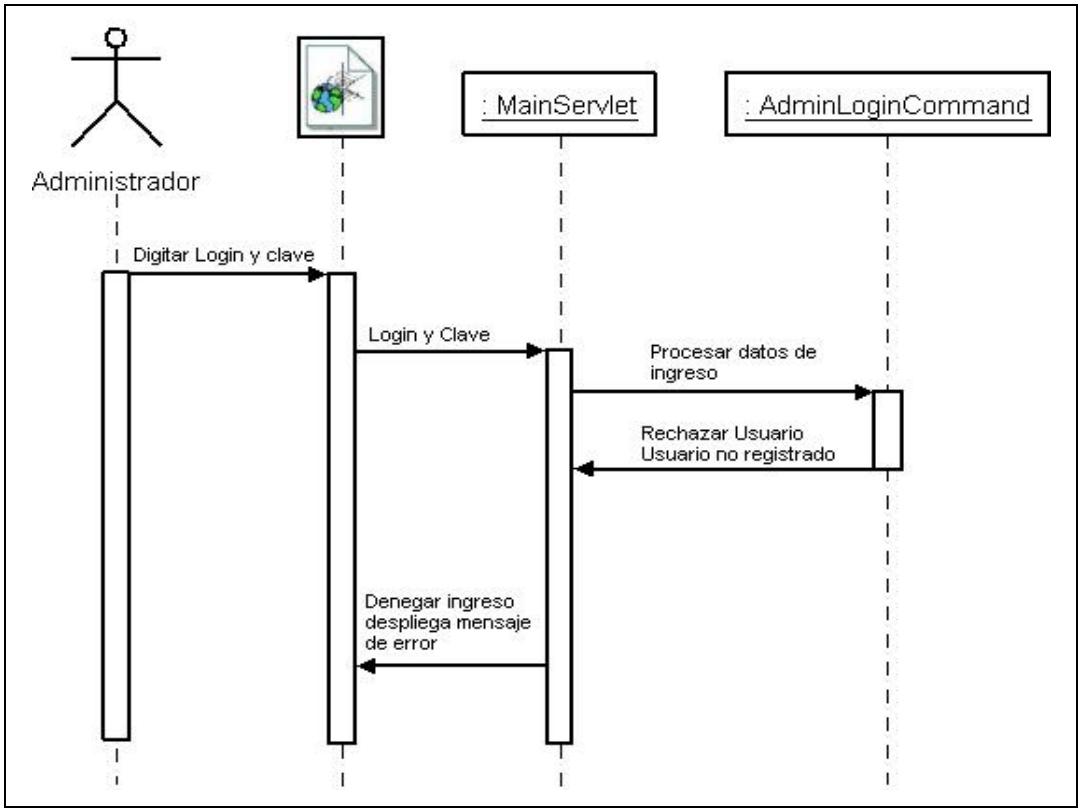

Figura 21. Diagrama caso de uso Ingresar al sistema Flujo Alternativo 1 (Administrador)

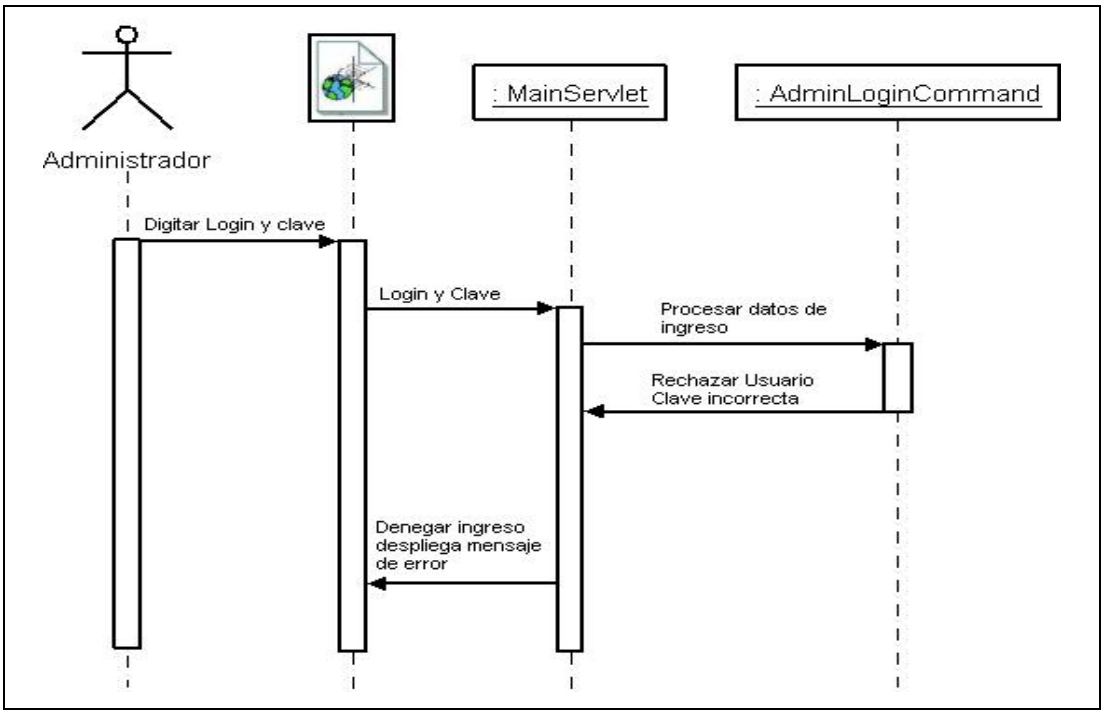

Figura 22. Diagrama caso de uso Ingresar al sistema Flujo Alternativo 2 (Administrador)

Las acciones relacionadas con el ingreso de nuevos usuarios, ya sea que el sistema lo ingrese o que no lo haga por verificar su existencia previa, la modificación de usuarios y eliminación de los mismos se pueden observar en las figuras de la 23 a la 26

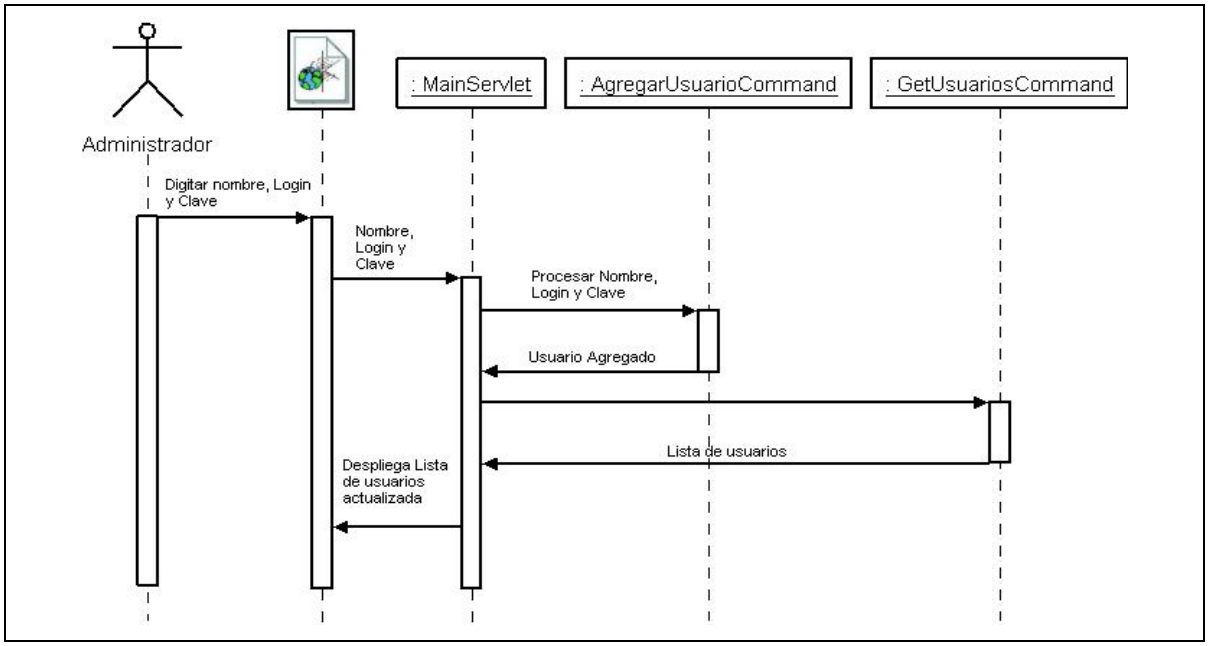

Figura 23. Diagrama del caso de uso Agregar Usuario Flujo Normal

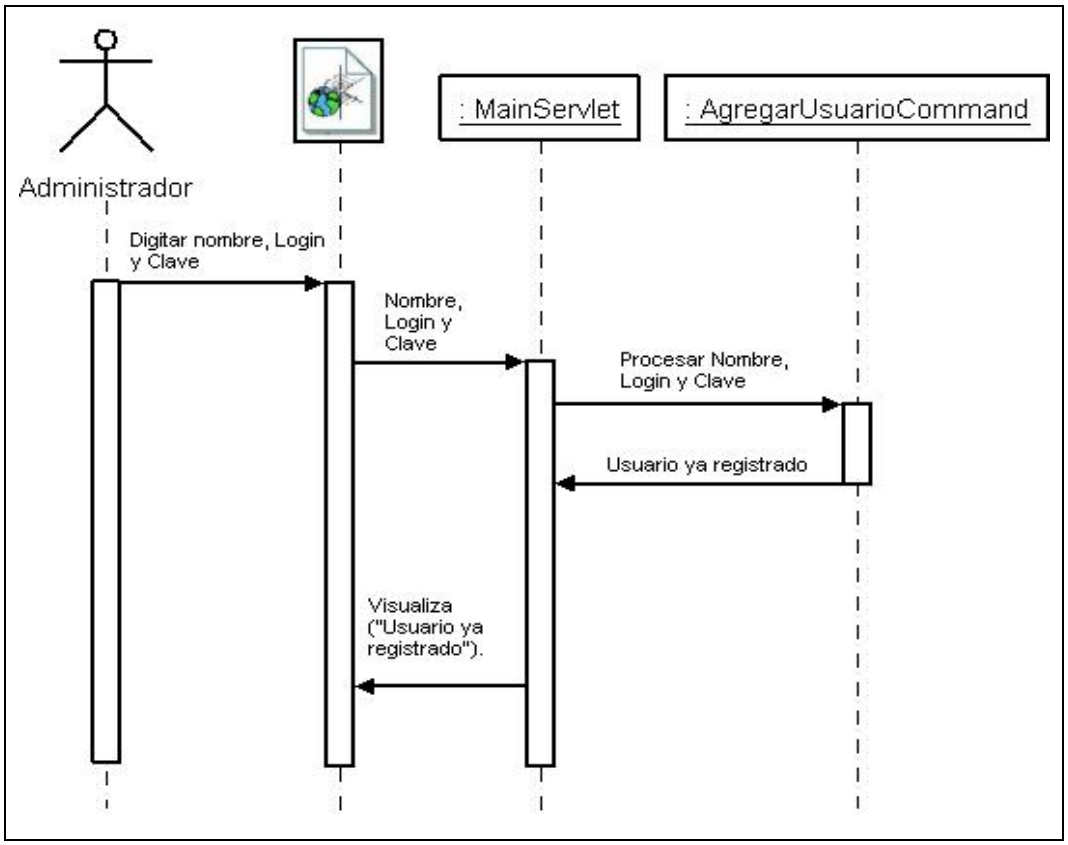

Figura 24. Diagrama del caso de uso Agregar Usuario Flujo Alternativo

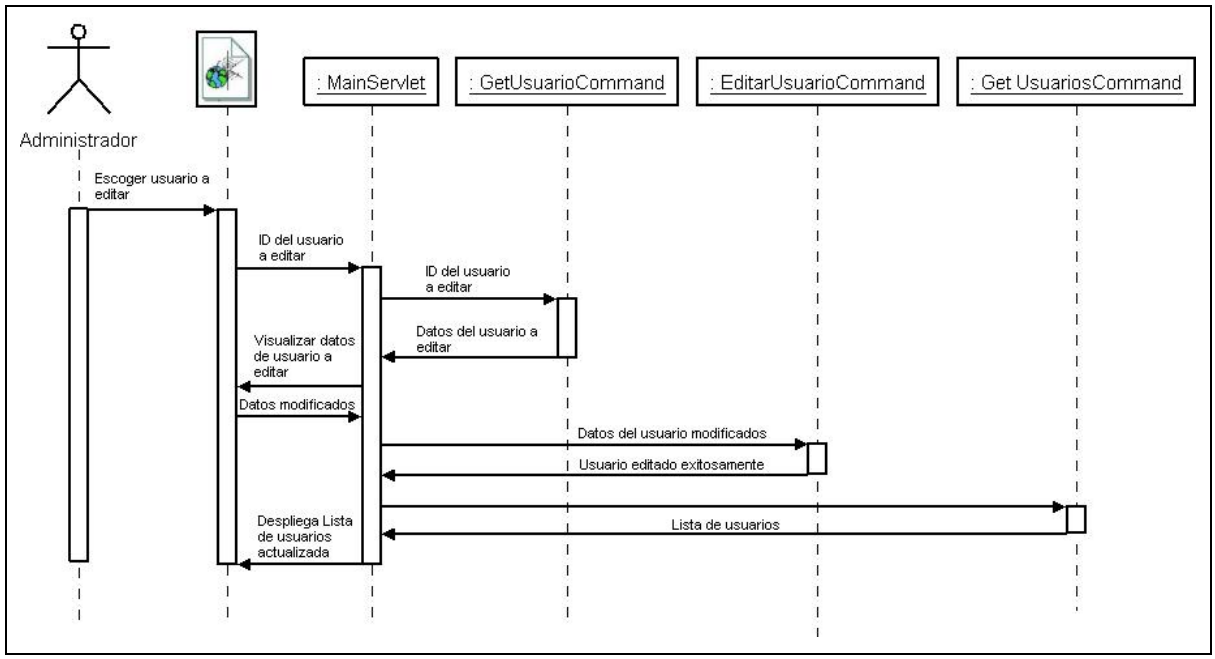

Figura 25. Diagrama del caso de uso Editar Usuario

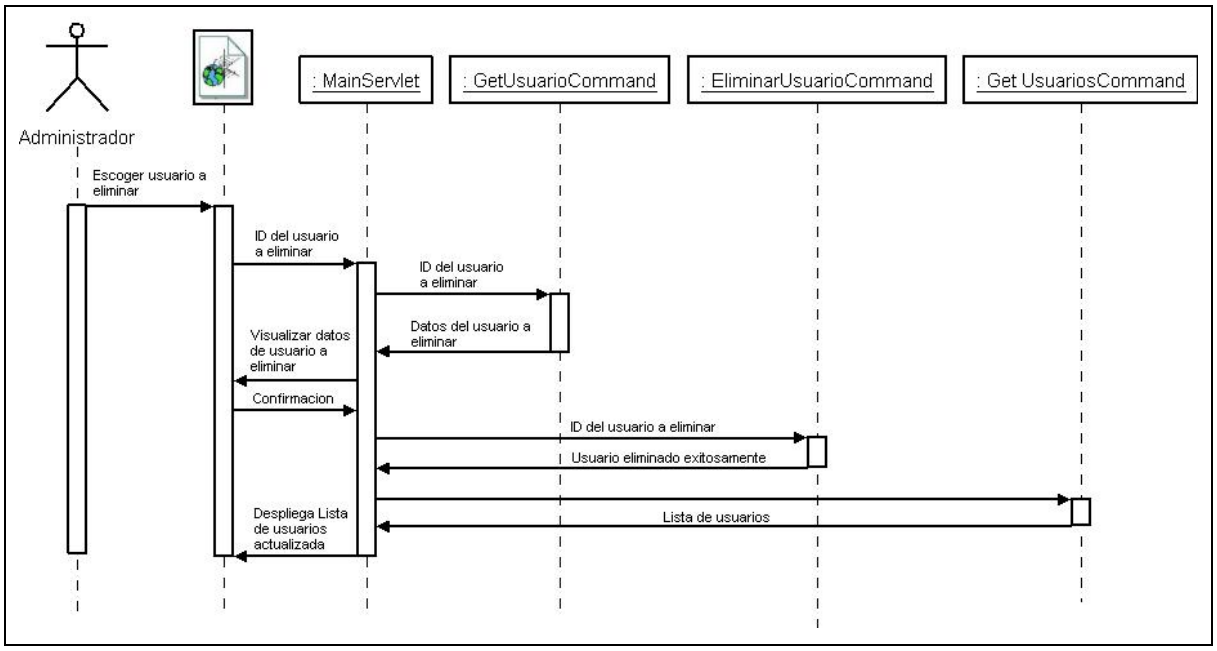

Figura 26. Diagrama del caso de uso Eliminar Usuario

El administrador se encarga también de las acciones relacionadas con el ingreso, modificación y eliminación de Sistemas de Bases de datos relacionales DBMS que soporta el sistema. Estas acciones se muestran en las figuras 27 a 29.

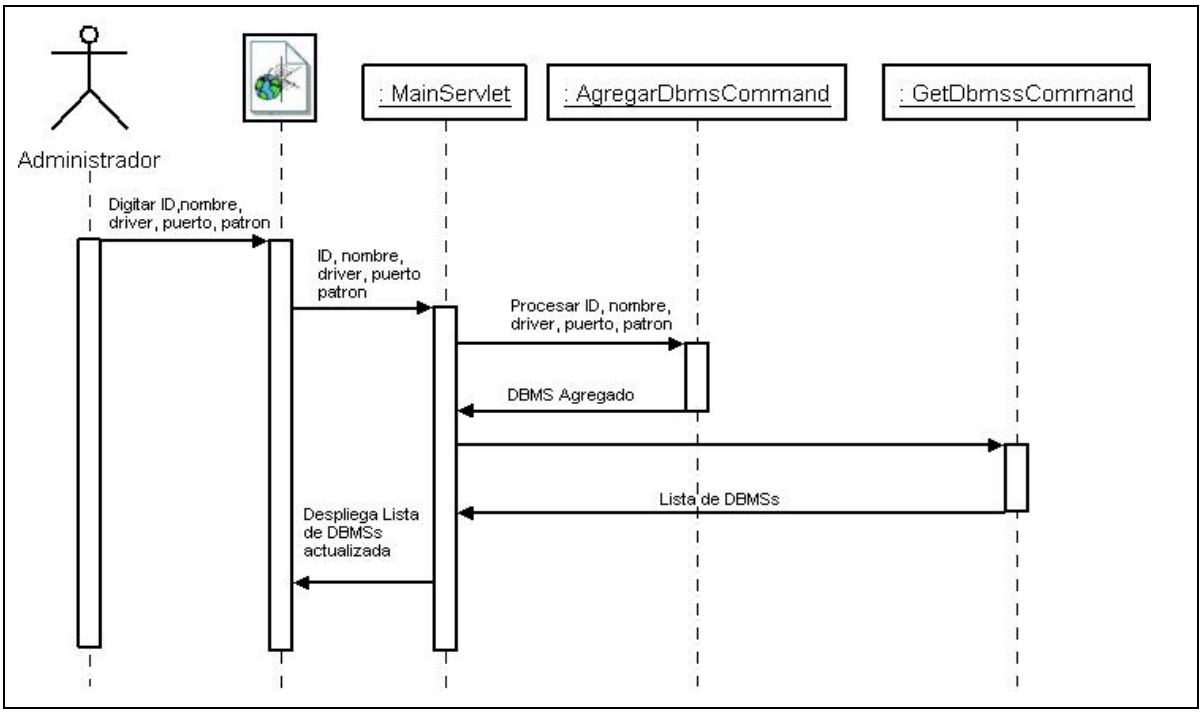

Figura 27. Diagrama del caso de uso Agregar DBMS

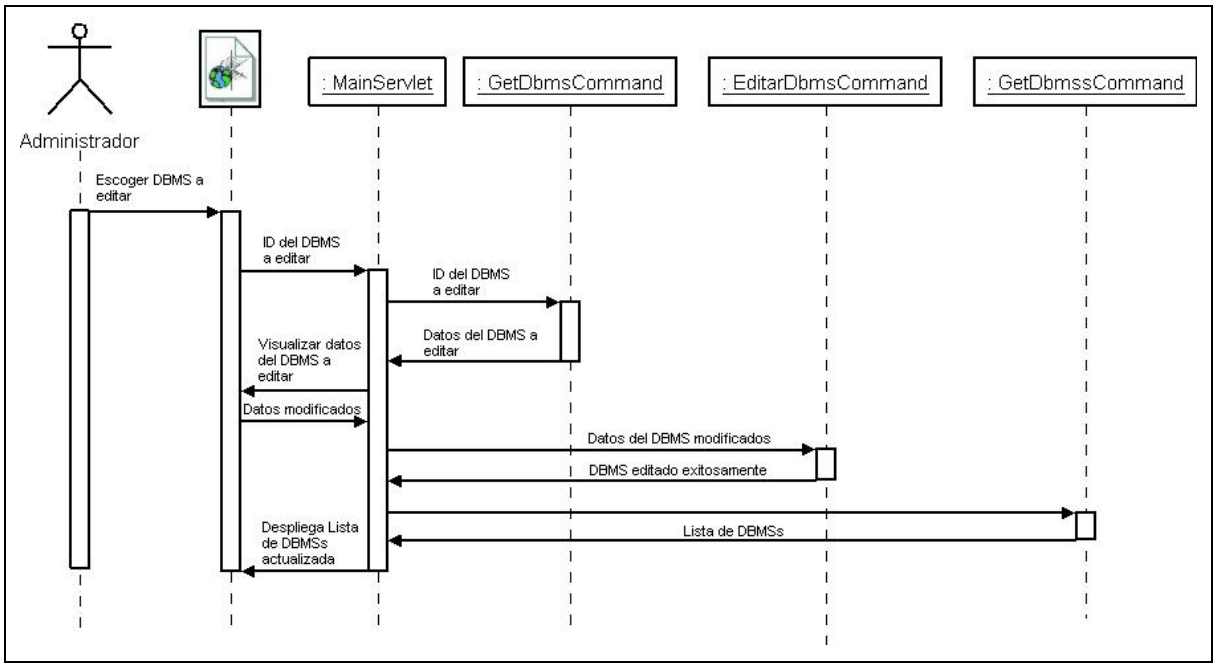

Figura 28. Diagrama del caso de uso Editar DBMS

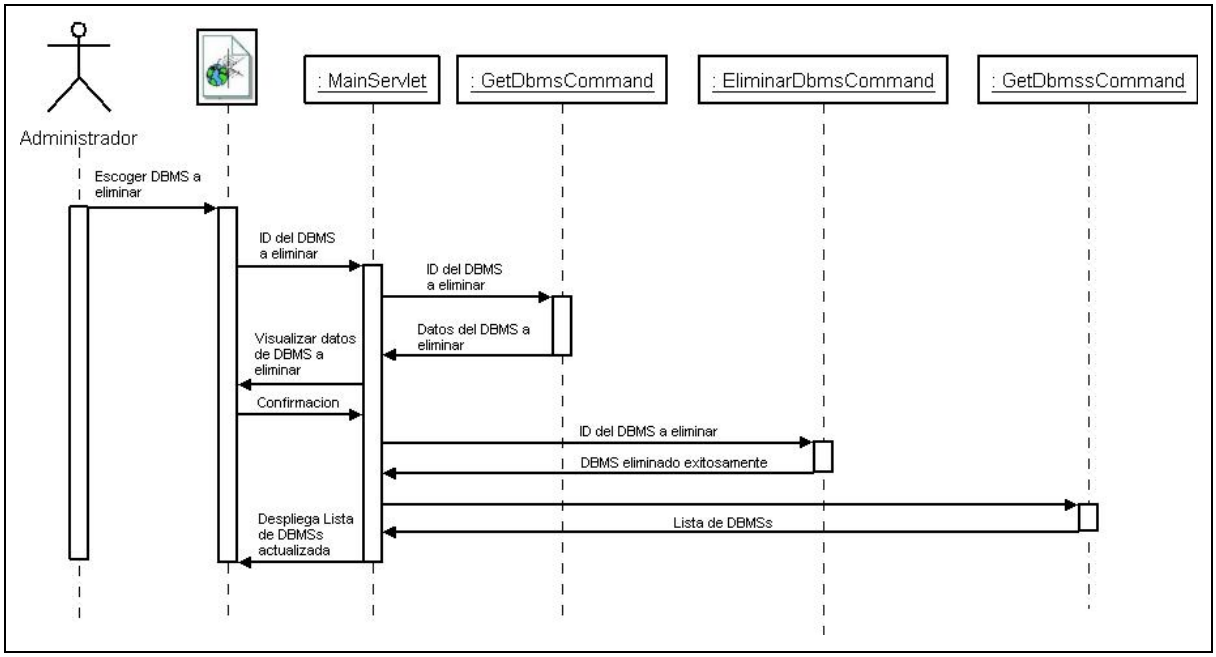

Figura 29. Diagrama del caso de uso Eliminar DBMS

Por último, el Administrador luego que realiza las operaciones sobre el sistema debe abandonarlo utilizando el mecanismo que el mismo ofrece para tal fin. Esta salida del sistema se muestra en la figura 30.

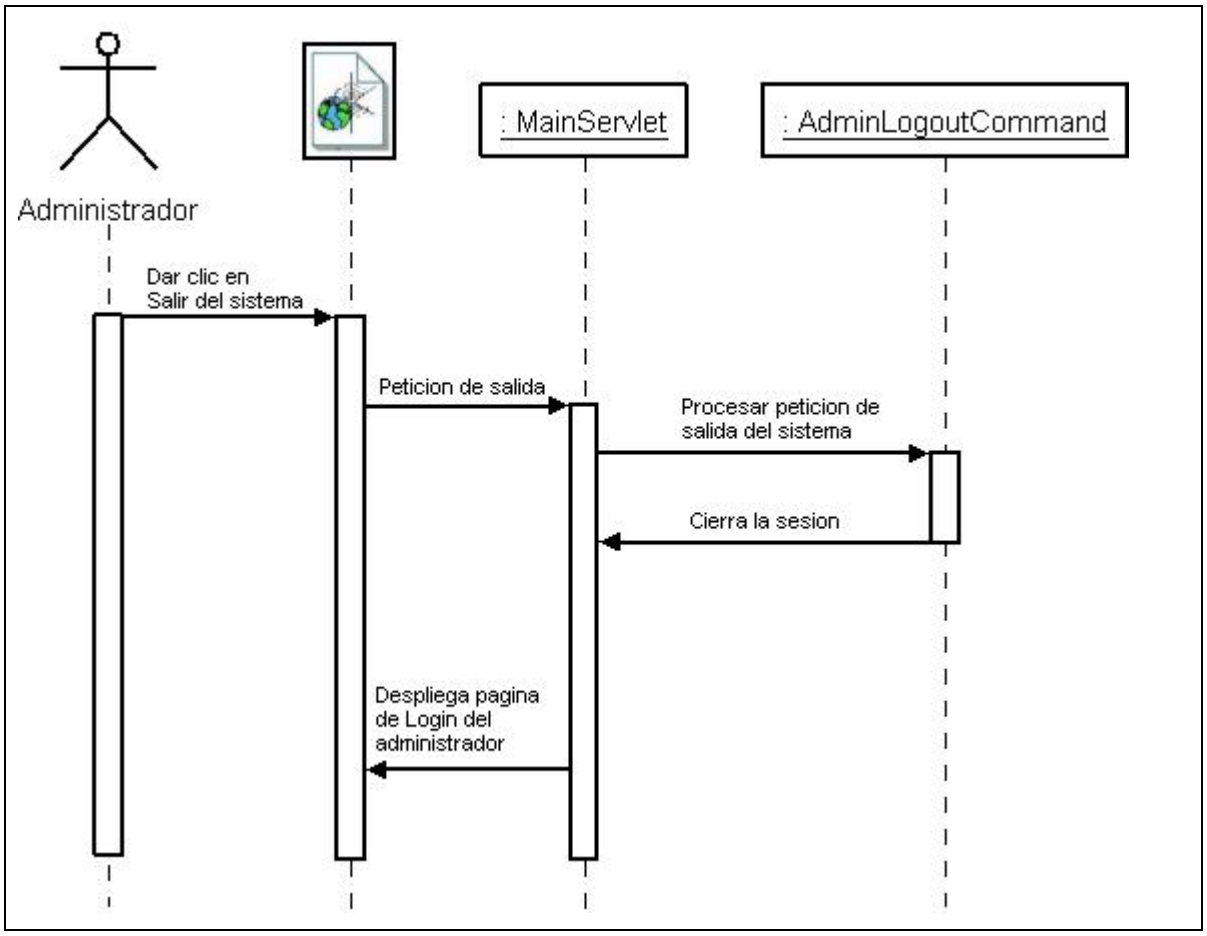

Figura 30. Diagrama del caso de uso Salir del sistema (Administrador)

A continuación mostraremos todos los procesos que puede realizar el actor Alumno sobre el sistema ya instalado y configurado previamente por el Administrador.

En primer lugar, el alumno debe ingresar al sistema colocando las credenciales adecuadas para su ingreso a la base de datos que elija. Este proceso se refleja en las figuras 31 y 32.

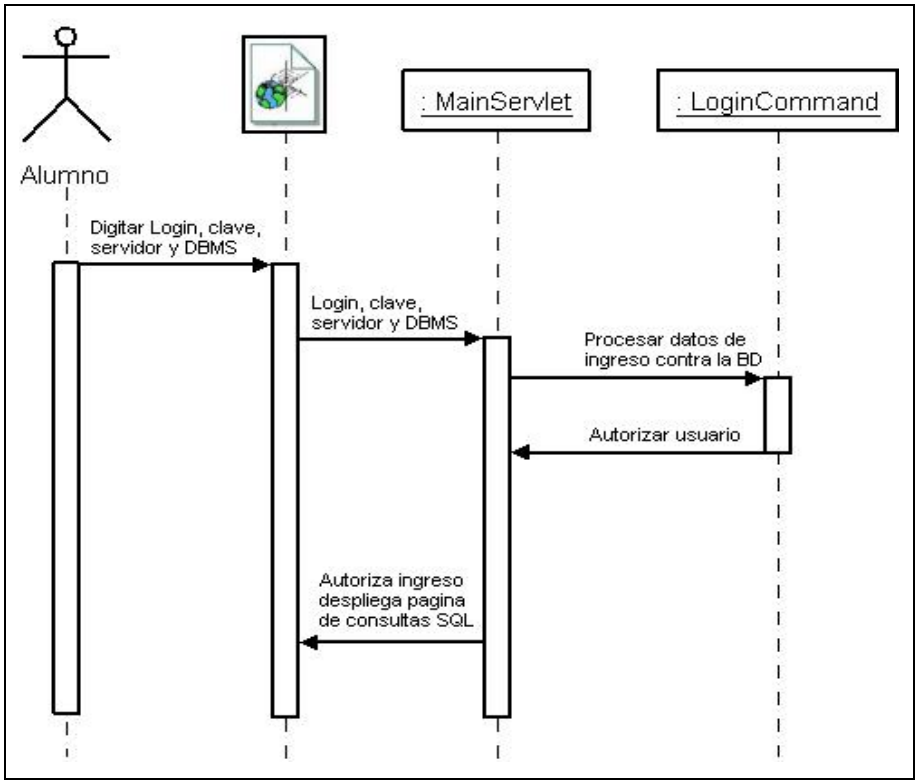

Figura 31. Diagrama del caso de uso Ingresar a la BD (Alumno) Flujo Normal

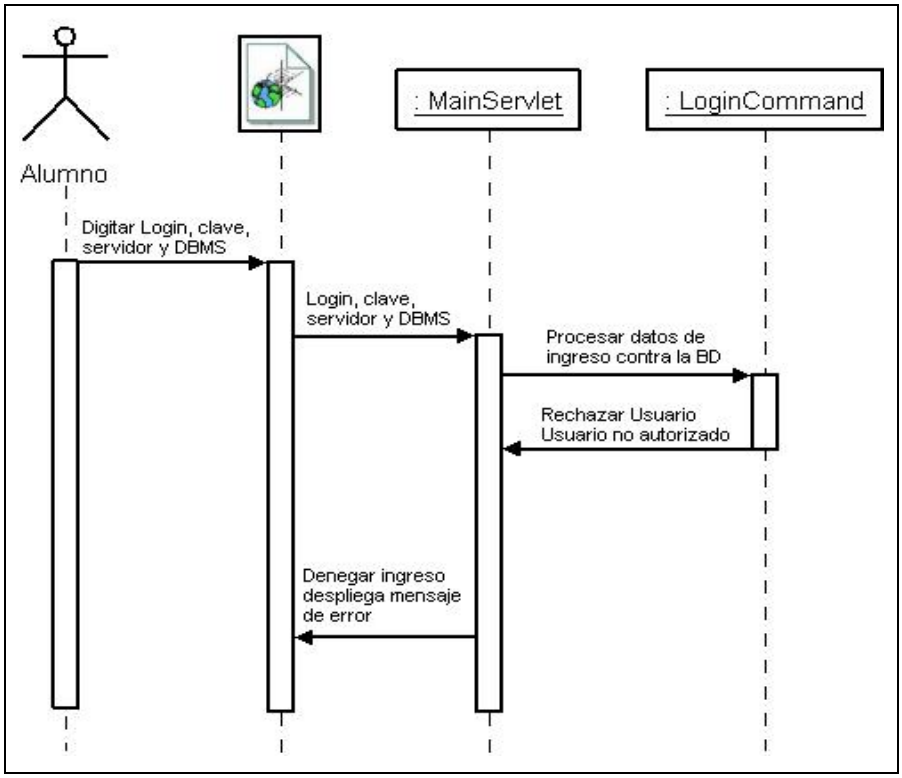

Figura 32. Diagrama del caso de uso Ingresar a la BD (Alumno) Flujo Alternativo

Una vez el Alumno ha ingresado al sistema, podrá ejecutar varias acciones sobre el mismo como son: Ejecutar sentencias SQL sobre el motor de bases de datos escogido (Figura 33 a 35) o verificar la sintaxis de una sentencia SQL (Figura 36 y 37).

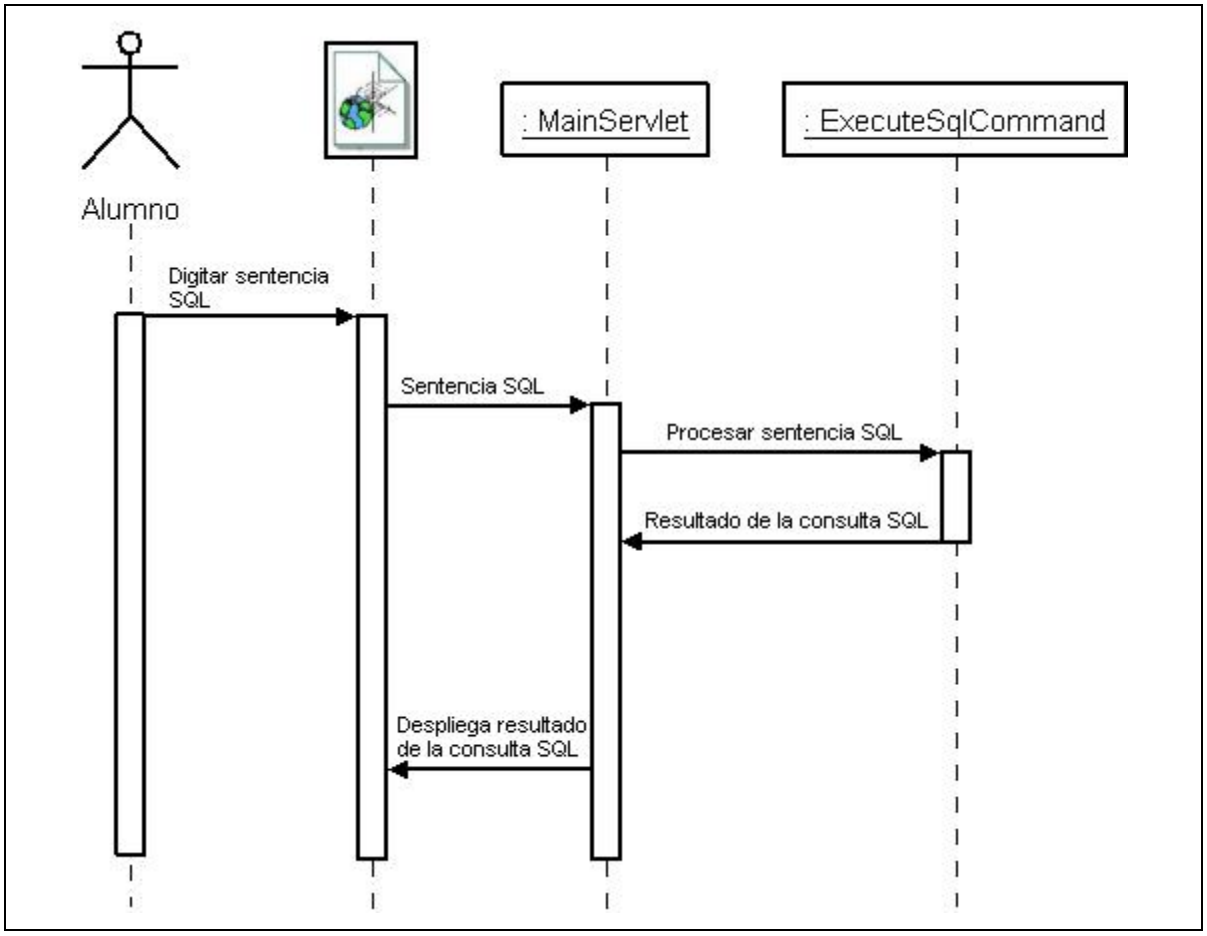

Figura 33. Diagrama del caso de uso Consultar BD Flujo Normal

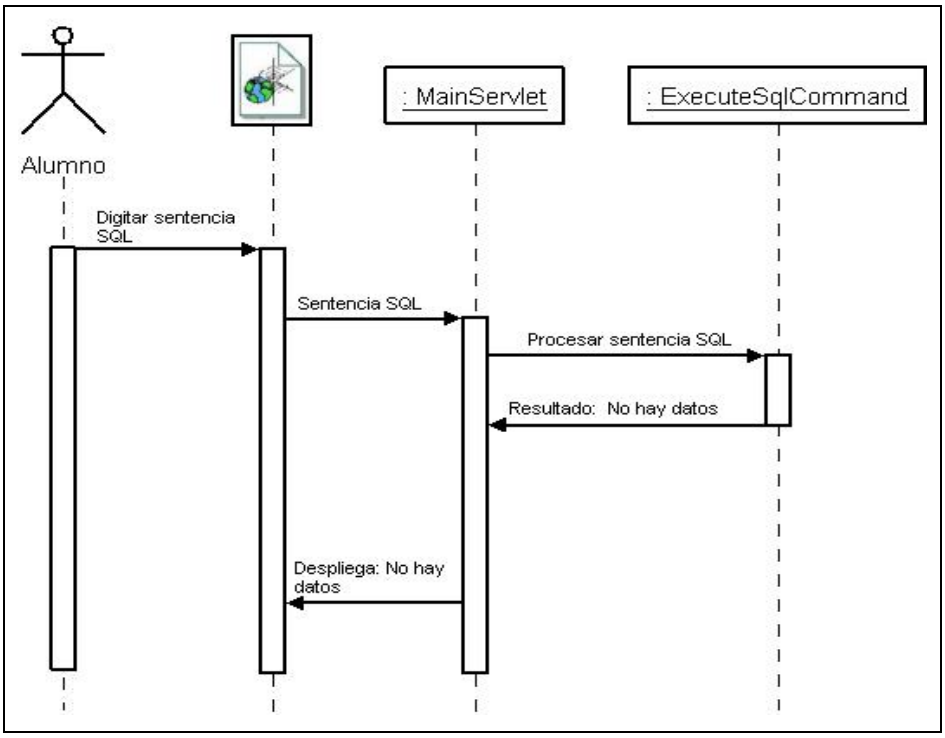

Figura 34. Diagrama del caso de uso Consultar BD Flujo Alternativo 1

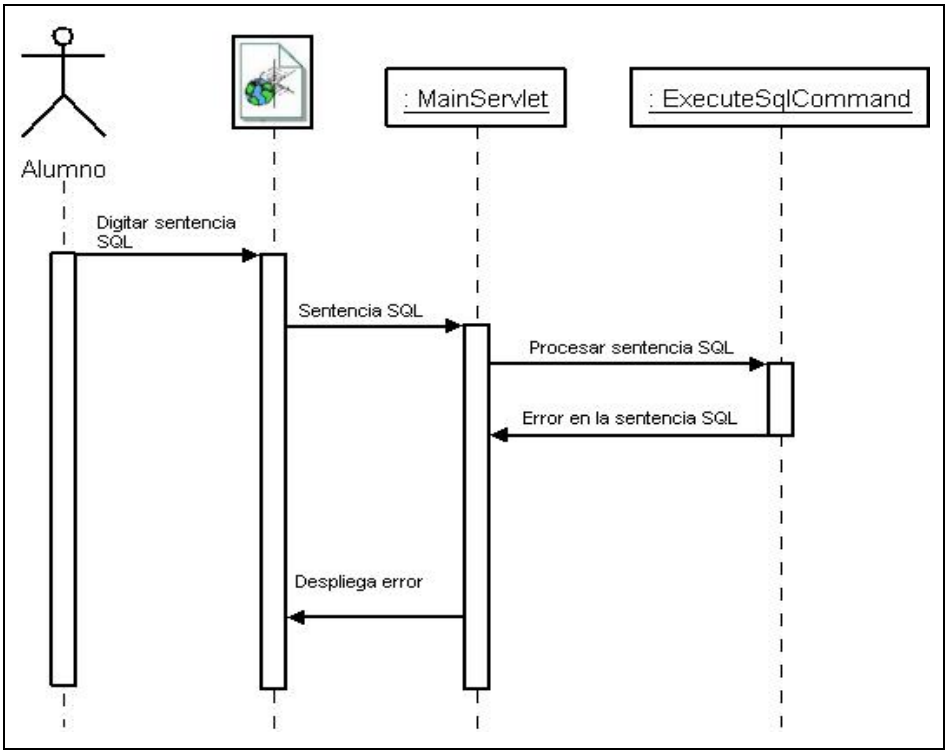

Figura 35. Diagrama del caso de uso Consultar BD Flujo Alternativo 2

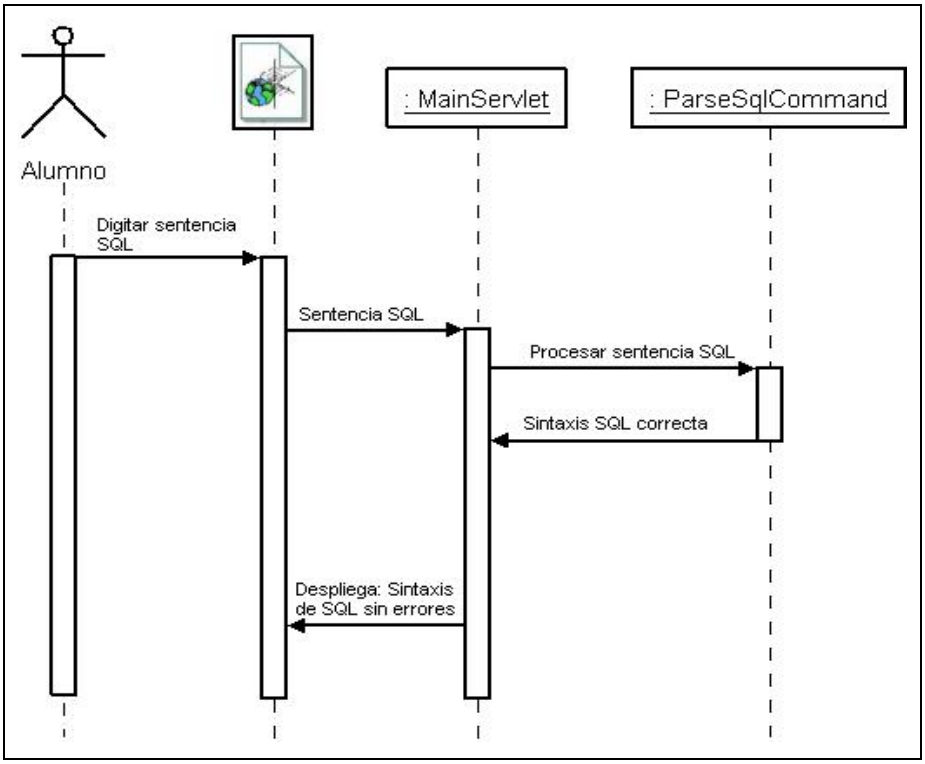

Figura 36. Diagrama del caso de uso Verificar Sintaxis Flujo Normal

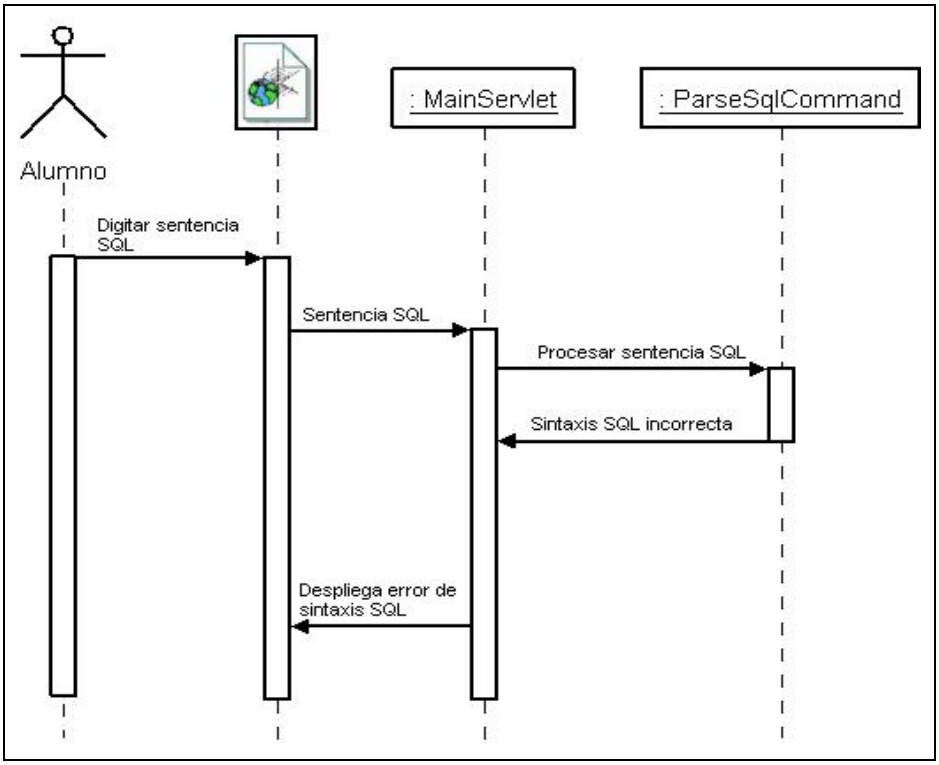

Figura 37. Diagrama del caso de uso Verificar Sintaxis Flujo Alternativo

Por último, el Alumno debe también salir del sistema una vez que termine de realizar sus tareas y lo debe realizar a través de la opción “Salir del Sistema” provista por la aplicación (Figura 38).

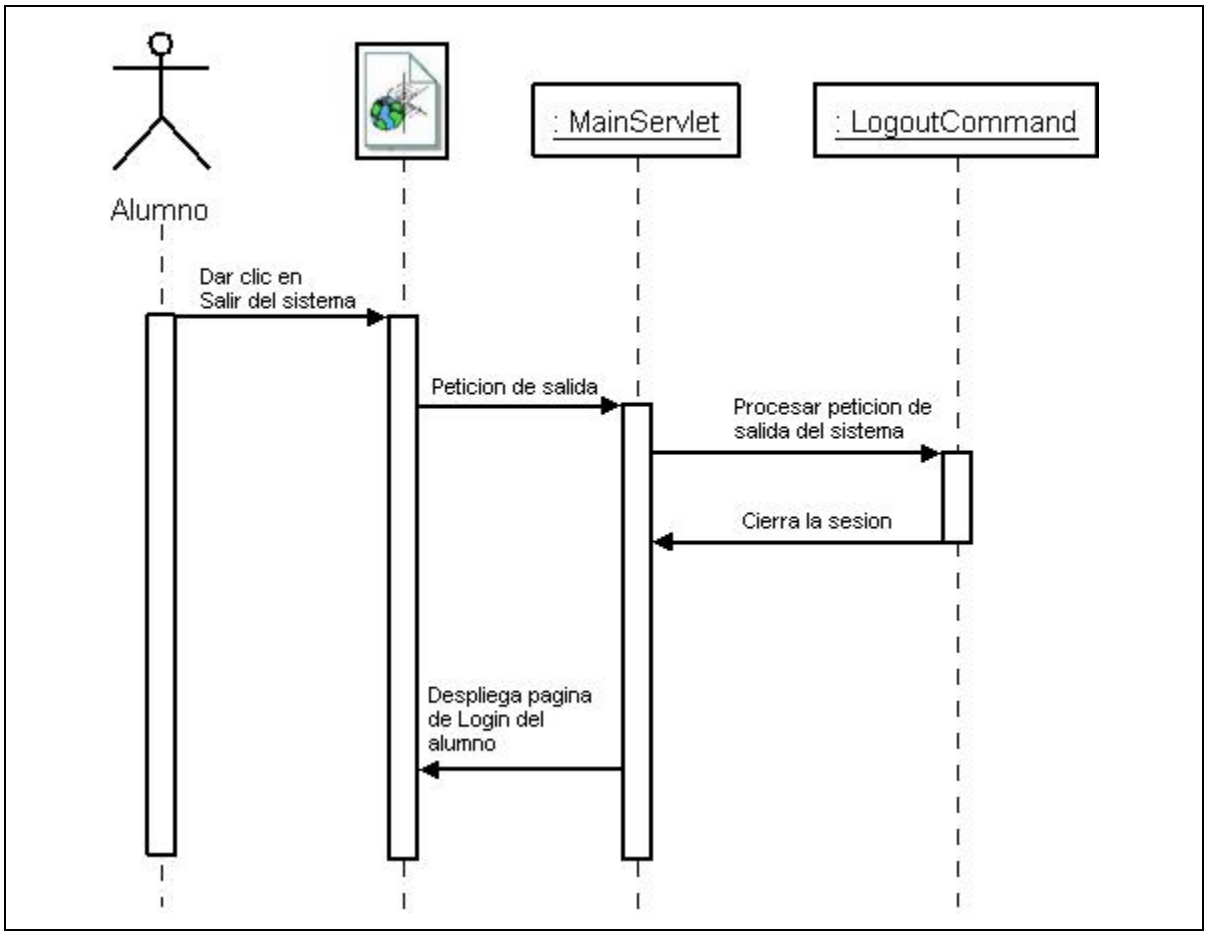

Figura 38. Diagrama del caso de uso Salir del Sistema (Alumno)

6.1.3 Diagramas de Clases

Los diseños a continuación muestran como interactúan las clases usadas para el desarrollo de nuestro sistema.

En la figura 39 mostramos la interacción de paquetes de la aplicación y las clases que conforman cada paquete y sus relaciones entre si.

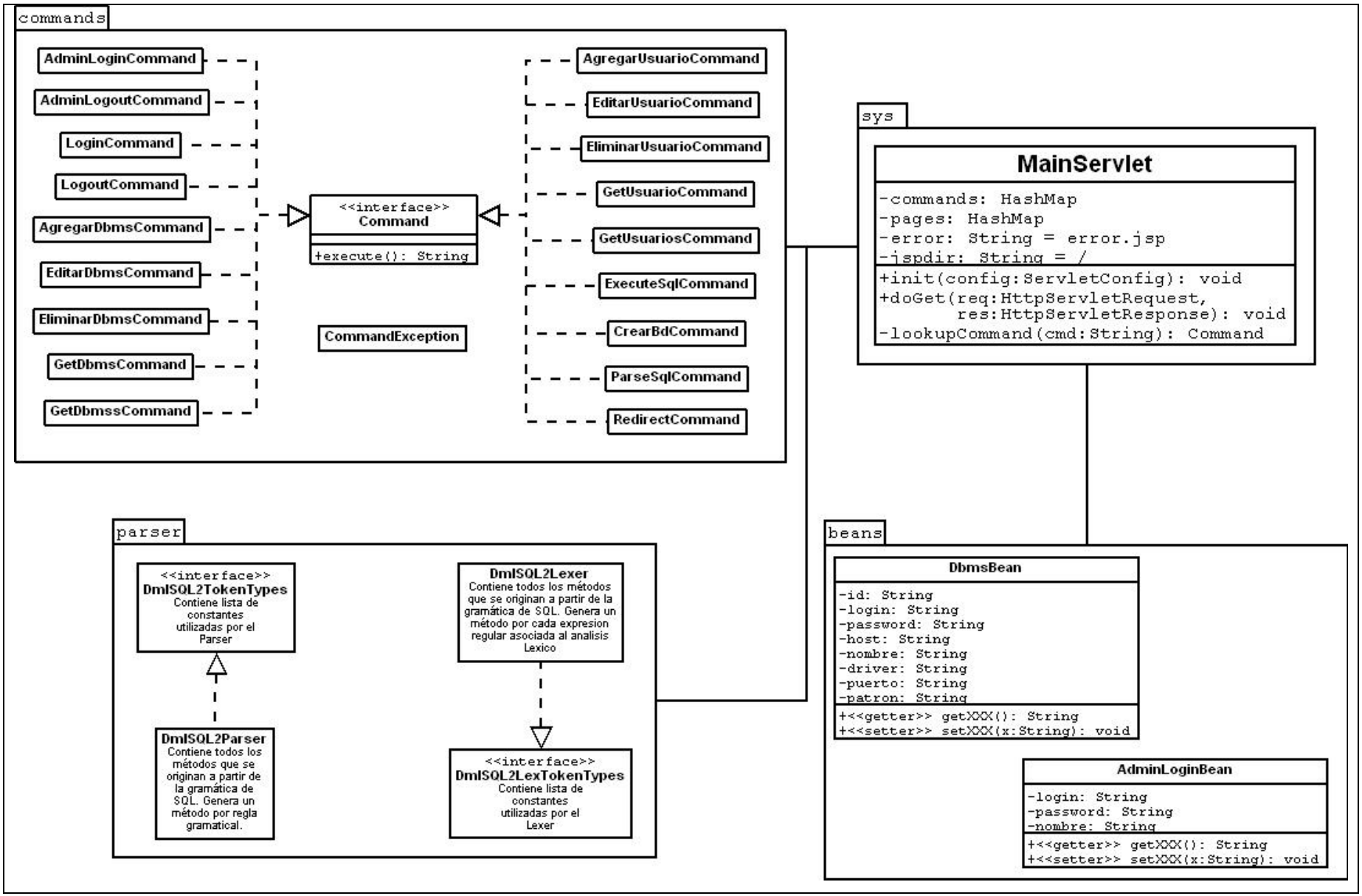

Figura 39. Diagrama de paquetes de la aplicación y sus diagramas de clases asociados

En la figura 40 mostramos el diagrama de clases completo con las relaciones entre clases sin importar el paquete en que se encuentran.

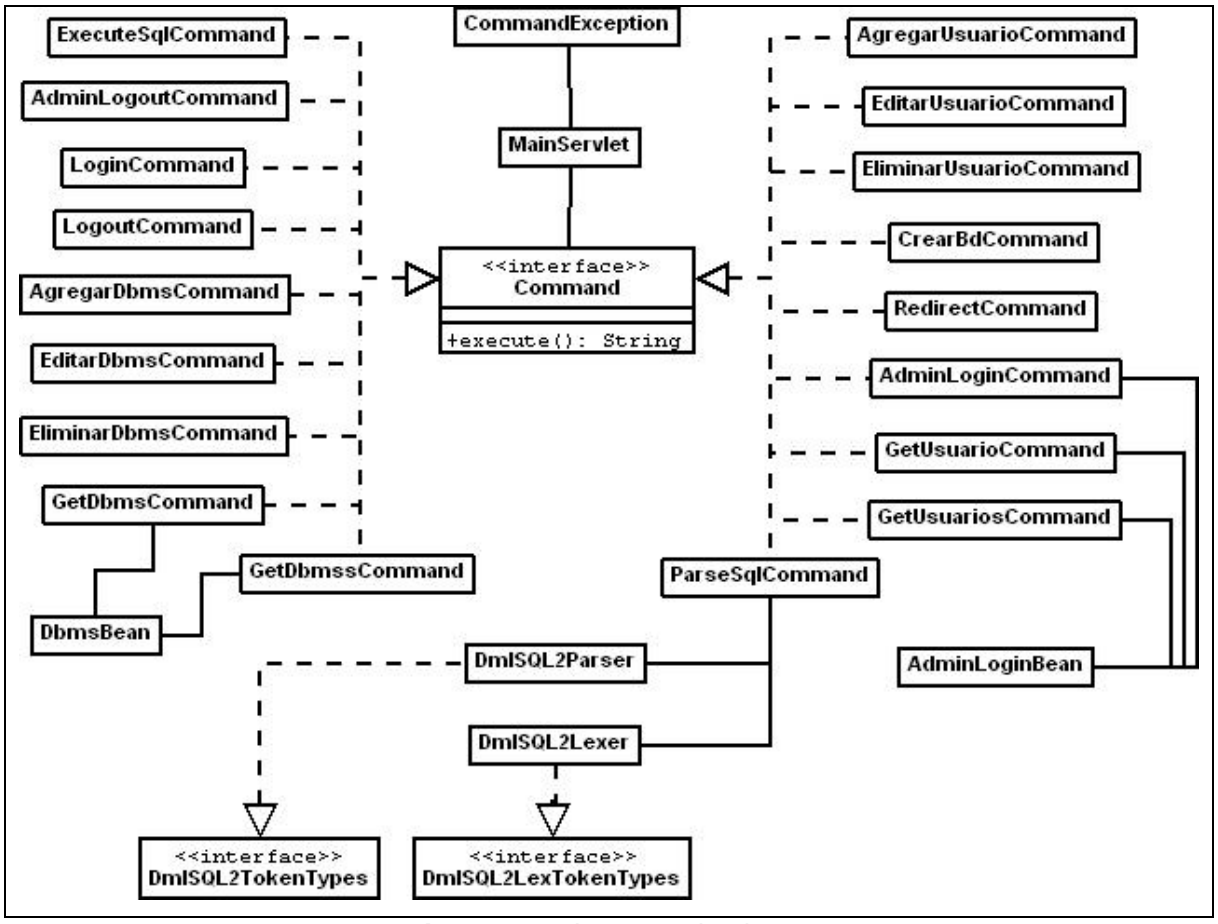

Figura 40. Diagrama de clases

6.2 COMPONENTES DEL DISEÑO MVC

A continuación se detalla más profundamente los componentes de la aplicación.

6.2.1 COMPONENTE MODEL

El componente Model es lo que define la lógica de la aplicación. Se subdivide a su vez en componentes de Estado y componentes de Acción. (ver figura 41)

Los componentes de Estado muestran el conjunto de valores actuales del modelo de datos e incluye métodos para cambiar dichos valores. La tecnología utilizada para la implementación de este tipo de componentes son los JavaBeans. La naturaleza reusable de los JavaBeans permite la implementación de los componentes de estado de manera independiente del protocolo de la aplicación.

Los componentes de Acción definen los cambios permitidos sobre el estado del modelo en respuesta a eventos que genera el usuario. La elección de la tecnología o de la capa donde se implementarán estos componentes no es tan sencilla; en una aplicación simple, es sencillo implementar las acciones en el mismo Controller, pero no se recomienda. Se crean entonces los Action Beans (Componentes de acción) los cuales si deben tener en cuenta el protocolo de la aplicación para poder determinar los eventos generados y la forma en que deben enviar la respuesta al usuario.

En la aplicación se generaron varios componentes de acción, que permiten entre otras cosas realizar las siguientes tareas, en respuesta a eventos generados por el usuario:

- Entrada al sistema: el usuario proporciona los datos necesarios para la conexión con alguna base de datos de un DBMS para el cual la aplicación tenga soporte. El componente de acción intentará establecer una conexión con los datos obtenidos y si esta es exitosa mostrará la página de ejecución de sentencias SQL, almacenando antes en la sesión de la aplicación el objeto de conexión para su utilización posterior dentro de ella. También se almacena en un componente de estado los parámetros de la conexión ingresados por el usuario para referencias futuras dentro de la aplicación.

- Verificación de sintaxis: se creó un componente de acción que realiza la verificación de la sintaxis de las sentencias DML del lenguaje SQL ANSI92 que el usuario escriba en la ventana del navegador, con la ayuda de una clase Helper (clase ayudante) que en este caso es el Parser generado con ANTLR a partir de la gramática del lenguaje. Al final de la verificación de sintaxis se crea un componente de estado (JavaBean) que le indicará a la aplicación si la sentencia fue verificada y si dicha verificación fue exitosa o no. Esto para que cuando se

mande a ejecutar la sentencia, no se repita el proceso de verificación en caso que ésta haya sido exitosa.

- Ejecución de sentencias SQL: se creó un componente que manda a ejecutar en el servidor al cual se está conectado, las sentencias SQL que el usuario digita. Si el usuario no realizó una verificación previa, este componente llamará al componente de acción encargado de verificar la sentencia (en caso que sea DML) para así evitar que el servidor se encargue del parseo de la misma. Si el usuario ya ha realizado la verificación, el sistema enviará la sentencia directamente al servidor para su ejecución y se retornará el resultado para su despliegue en pantalla, si dicha ejecución fue exitosa, o una excepción en caso contrario.

- Salida del sistema: se creó un componente que invalida la sesión actual de la aplicación para destruir todos los objetos almacenados en ella como JavaBeans, objetos de conexión, y otros más dejando al usuario en la pantalla inicial de la aplicación para que este pueda ingresar datos adicionales para conexión con otra base de datos.

6.2.2 COMPONENTE VIEW

El componente View de la aplicación está conformado por todos los archivos HTML, CSS, JavaScript y JSP que presentan la información en el navegador del usuario. (ver figura 41) Los archivos JSP son los encargados de recibir la información que envía el Model a través del Controller y establecer la manera de presentarlos al usuario. También se encargan de desplegar la interfaz de interacción del usuario con la aplicación con la ayuda de archivos estáticos HTML, de hojas de estilo CSS y de JavaScript para generar efectos de animación y/o validación de las entradas por parte del usuario.

La separación del componente View del componente Model permite la implementación de interfaces de usuario con diferente Look and Feel. Todas estas interfaces interactúan con el mismo Model.

En la aplicación se crearon archivos JSP para la entrada al sistema, para la presentación de los mensajes de error debido a excepciones de la aplicación, para la entrada de las sentencias SQL y la posterior visualización de sus resultados. Además se crearon vistas para las páginas de administración de la aplicación donde se piden los datos de los DBMS que se van a agregar o donde se muestran los DBMS existentes que la aplicación soporta para su edición o eliminación.

6.2.3 COMPONENTE CONTROLLER

Este componente es que enlaza los otros dos elementos del patrón de diseño MVC. Es el responsable de recibir los eventos, determinar el manejador apropiado, invocar dicho manejador y finalmente, lanzar la generación de la respuesta apropiada. (ver figura 41)

Sabiendo todo el poder que da el lenguaje Java y sus tecnologías, los Servlets son la mejor elección para la implementación de este componente.

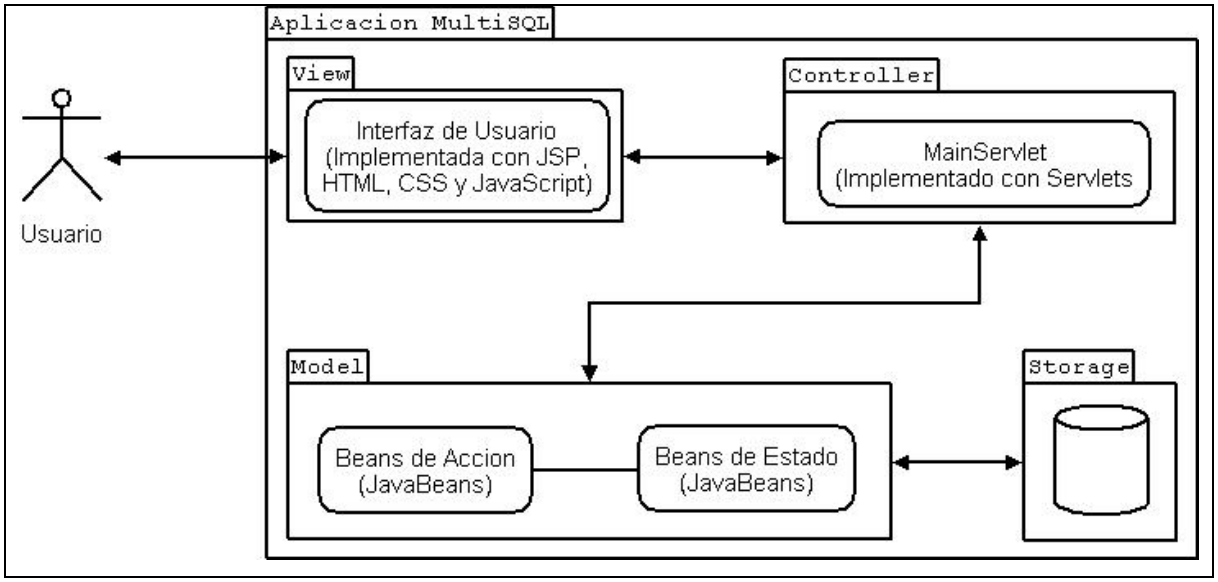

Figura 41. Modelo MVC de la aplicación MultiSQL

6.3 VENTAJAS DEL DISEÑO

Las ventajas de este desarrollo desde el punto de vista del diseño; son entre otras, lo constituyen las mismas ventajas reportadas como ventajas de desarrollar aplicaciones

Cliente/Servidor con el modelo MVC es decir básicamente la segregación en capas para las piezas de software que facilita el mantenimiento e incrementa la facilidad de introducir mejoras funcionales al sistema (Detalladas en la sección 4.2.5).

Todo esto sin tener en cuenta las ventajas inherentes de fusionar en el mismo proyecto metodologías de desarrollo (Modelo Cliente/Servidor Multicapa), herramientas avanzadas de generación de parsers (ANTLR), y Programación Orientada a Objetos y tecnologías como: Java, Servlets, JSP y Javabeans basados en Java cuyas ventajas se mencionan en la documentación incluida en las secciones 5.1.2, 5.1.3 y 5.3.1.

6.4 ELEMENTOS DE LA IMPLEMENTACIÓN

La aplicación MultiSQL está implementada siguiendo el patrón de diseño de aplicaciones MVC (Modelo-Vista-Controlador) el cual nos provee niveles de abstracción total en cada uno de dichos niveles, permitiendo así poder modificar nuestro modelo de datos sin tener que modificar las interfaces de visualización y viceversa.

6.4.1 Componente Controlador (CONTROLLER)

El componente Controller de MultiSQL está implementada utilizando la tecnología de Servlets de J2EE, y está diseñado para ser el punto de entrada de todas las peticiones que recibe la aplicación. Este componente, como lo dije anteriormente, es un Servlet (MainServlet) que se encarga de recibir todas las peticiones originadas en la capa de Vista de la aplicación, con la cual interactúa el usuario, decide que Objeto de la capa del Modelo debe ejecutar la acción sobre los datos enviados y luego dirige la salida respectiva hacia la página correspondiente.

6.4.1 Componente Vista (VIEW)

Todas las páginas que componen la interfaz de usuario de la aplicación MultiSQL, están escritas utilizando tecnología JSP (Java Server Pages) de J2EE, la cual nos permite mezclar en el código HTML, código Java que va a tomar los datos recibidos desde la aplicación y los mostrará utilizando el formato que se desee al usuario.

6.4.2 Componente Model (DEL)

El componente Model de la aplicación MultiSQL es donde se localiza toda la “lógica de negocios” de nuestra aplicación. Está conformada por clases Java que se encargan de obtener, procesar y almacenar información para luego ser visualizada en la capa Vista por el usuario final. En esta capa tenemos varios tipos de objetos: Objetos independientes y JavaBeans. Los Objetos independientes se encargan de realizar labores secundarias y de apoyo para los demás objetos de la aplicación. Un ejemplo de estos objetos es HelperClass.

Los JavaBeans son los objetos principales que conforman esta capa de la aplicación, y están subdivididos en dos clases: Los Beans de acción y los Beans de estado.

Los Beans de acción, como su nombre lo indica, son los encargados de efectuar todas las operaciones de procesamiento de los datos, además de encargarse de su obtención desde la Base de Datos respectiva.

Los Beans de estado son los encargados de almacenar la información procesada por los Beans de acción y la transportarán hacia la capa de Vista para luego ser visualizados por el usuario.

El proceso de interacción del usuario con la aplicación implica los siguientes pasos concernientes al diseño de las distintas capas de la aplicación:

1. Cada página jsp que envíe datos para ser procesados por la aplicación, contiene dos campos de formulario que definen quien ejecuta el procesamiento de los datos y la página que luego visualizará el resultado de dicho procesamiento. Estos campos se llaman respectivamente “cmd” y “redirect”:

```
<form name="formEditarUsuario" method="POST" action="MainServlet">
<input type="hidden" name="cmd" value="EDITAR_USUARIO"/>
<input type="hidden" name="redirect" value="ADMIN_USUARIOS"/>
        ...
</form>
```

2. El controlador (MainServlet) recibe todos los datos enviados por el usuario, además de los campos del numeral anterior y luego busca en una tabla la clase correspondiente al código asignado para ejecutar el procesamiento sobre dichos datos, luego instancia un objeto de esta clase y le envía los datos para ser procesados.

3. Una vez el objeto (Bean de acción) recibe los datos para procesarlos (sea una consulta SQL, una petición para consultar un usuario del sistema, o una petición para agregar un nuevo motor de Base de Datos al sistema), este ejecuta su código, y luego determina, basado en el parámetro “redirect” hacia donde enviar el resultado del procesamiento realizado sobre los datos para que el usuario los visualice, esto es, determina el nombre de la página .jsp encargada de esta visualización.

4. Este resultado nuevamente le llega al Controlador el cual, ya con la información procesada y con el nombre de la página que va a visualizar el resultado, redirecciona la salida hacia dicha página.

5. La página JSP que recibe la información procesada, se encarga de formatearla para ser mostrada de la manera más óptima para el usuario quien luego de visualizarla, podrá nuevamente iniciar una nueva petición, la cual seguirá el mismo proceso aquí descrito.

La aplicación MultiSQL está subdividida en dos módulos principales. El módulo de Consultas y el módulo de Administración.

En el módulo de Consultas participan las siguientes clases y páginas JSP para llevar a cabo las tareas de consulta SQL en las bases de datos:

Páginas de la capa de Vista

index.jsp
header.html
internalheader.html
sql.jsp
error.jsp

Clases de la capa Modelo

DmlSQL2Lexer
DmlSQL2LexTokenTypes
DmlSQL2Parser
DmlSQL2TokenTypes
ExecuteSqlCommand
LoginCommand
LogoutCommand
ParseSqlCommand

En el módulo de Administración participan las siguientes clases y páginas JSP para llevar a cabo las tareas administrativas sobre el sistema como son, agregar usuarios, editarlos y elminarlos, agregar soporte para otros motores de bases de datos, lo mismo que editarlos o eliminarlos:

Páginas de la capa Vista:

admin.jsp
internalheaderadmin.html
adminLogin.jsp
adminUsuarios.jsp
adminDbmss.jsp
agregarUsuario.jsp
editarUsuario.jsp
eliminarUsuario.jsp
agregarDbms.jsp
editarDbms.jsp
eliminarDbms.jsp

Clases de la capa Modelo:

AdminLoginBean
AdminLoginCommand
AdminLogoutCommand
AgregarDbmsCommand
EditarDbmsCommand
EliminarDbmsCommand
DbmsBean
AgregarUsuarioCommand
EditarUsuarioCommand
EliminarUsuarioCommand
GetDbmsCommand
GetDbmssCommand
GetUsuarioCommand
GetUsuariosCommand

7. IMPLEMENTACIÓN DEL SOFTWARE

La aplicación MultiSQL fue implementada enteramente usando tecnologías estándar (Java Servlets, Java Server Pages, MVC) y herramientas de uso libre (J2SE SDK, Apache Tomcat, Apache Ant, ANTLR). Se definieron los componentes de la aplicación siguiendo la arquitectura MVC y explicará a continuación cada una de ellas.

El controlador de la aplicación se implementó usando un Servlet. El método init() del servlet se usó para inicializar los parámetros necesarios para procesar los requerimientos del cliente, como son los manejadores de eventos.

```
public void init(ServletConfig config) throws ServletException {
        super.init(config);
        ResourceBundle rbCommands =
ResourceBundle.getBundle("Commands");
        Enumeration eCmds = rbCommands.getKeys();

        commands = new HashMap();
        while (eCmds.hasMoreElements()) {
            String key = (String) eCmds.nextElement();
            String value = rbCommands.getString(key);

            try {
                Command command = (Command)
Class.forName(value).newInstance();
                commands.put(key, command);
            }
            catch (Exception ex) {
            }
```

```
        }
    }
```

El método doGet() llama al método doPost(), lo cual garantiza que sea cual fuere el método de paso de parámetros utilizado por la vista, el Servlet responderá de la misma manera.

Se Implementó la manera de localizar el manejador de eventos utilizando un HashMap para almacenar todos los nombres de eventos conocidos (la llave) y su correspondiente clase manejadora de evento (el evento en sí). Cada manejador de evento debe implementar la acción para procesar el evento y deberá entregar una respuesta y un elemento de la vista para que el Controller pueda hacer el direccionamiento a la misma y visualizar la información.

```
public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        doPost(req, res);
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        String next;

        try {
            Command command = lookupCommand(req.getParameter("cmd"));
            next = command.execute(req);}
        catch (CommandException e) {
            req.setAttribute("javax.servlet.jsp.jspException", e);
            next = this.error;}

        RequestDispatcher rd;
        rd = getServletContext().getRequestDispatcher(this.jspdir + next);
```

```
            rd.forward(req, res);}

        private Command lookupCommand(String cmd) throws CommandException {
            if (cmd == null)
                cmd = "LOGIN";
            if (commands.containsKey(cmd))
                return (Command) commands.get(cmd);
            else
                throw new CommandException(cmd + " - Invalid Command
Identifier");}
```

El archivo de propiedades Commands.properties es la entrada para cargar los nombres de evento con su correspondiente manejador de eventos.

```
LOGIN=commands.LoginCommand
LOGOUT=commands.LogoutCommand
EXECSQL=commands.ExecuteSqlCommand
PARSESQL=commands.ParseSqlCommand
```

El componente Model de la aplicación, donde está toda la lógica de la misma se implementó por medio de componentes de Acción que son lanzados por el Controller y componentes de estado implementados como JavaBeans.

Se diseñó una interfaz genérica para definir el comportamiento de todos los manejadores de evento, que indica que todos y cada uno de ellos tienen que implementar un método llamado execute() y que deben devolver como respuesta una cadena de caracteres que contendrá la página de la Vista a donde se dirigirá la salida para presentarla al usuario.

```
package commands;
```

```
import javax.servlet.*;
import javax.servlet.http.*;

public interface Command {public String execute(HttpServletRequest req) throws CommandException;}
```

Cada manejador de evento deberá entonces implementar esta interfaz, es decir, copiar su comportamiento. Se definieron manejadores de evento para cada una de las opciones que tiene la aplicación:

Evento	Manejador
Inicio de Sesión	commands.LoginCommand
Salida del Sistema	commands.LogoutCommand
Verificación de sintaxis	commands.ParseSqlCommand
Ejecución de sentencias SQL	commands.ExecuteSqlCommand
Agregar soporte para un DBMS al sistema	commands.AddDbmsCommand
Editar propiedades de un DBMS soportado	commands.EditDbmsCommand
Eliminar soporte para un DBMS del sistema	commands.DeleteDbmsCommand

Tabla 8. Eventos

A continuación se muestra uno de estos manejadores de eventos para ilustrar su implementación:

```
package commands;

import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;
```

```
import java.text.*;

public class LoginCommand implements Command {
	public String execute(HttpServletRequest req) throws CommandException {
		String nextPage = "";
		ResourceBundle rbPages = ResourceBundle.getBundle("Pages");
		ResourceBundle rbPatrones = ResourceBundle.getBundle("Patrones");
		ResourceBundle rbPorts = ResourceBundle.getBundle("Ports");
		ResourceBundle rbDrivers = ResourceBundle.getBundle("Drivers");

		Connection con = null;

		String host = req.getParameter("host");
		String db = req.getParameter("db");
		String userName = req.getParameter("usuario");
		String password = req.getParameter("clave");
		String dbms = req.getParameter("dbms");
		String patron = rbPatrones.getString(dbms);
		String port = rbPorts.getString(dbms);
		String driver = rbDrivers.getString(dbms);

		String args[] = { new String(host), new String(port), new String(db) };

		String conString = MessageFormat.format(patron, args);

		try {
			Class.forName(driver);
			con = DriverManager.getConnection(conString, userName,
password);
			if (con == null) {
				nextPage = "index.jsp";
				req.setAttribute("msg", "Usuario no autorizado");}
			else {
```

```
                    nextPage = rbPages.getString(req.getParameter("redirect"));
                    req.setAttribute("msg", null);
                    HttpSession session = req.getSession();
                    session.setAttribute("connection", con);}}
            catch (Exception e) {}
            return nextPage;}}
```

Este manejador de eventos se encarga de dar la entrada al usuario al sistema, realizando la conexión con la base de datos que el usuario especifique. Para esto, toma de los archivos de configuración toda la información referente al motor de bases de datos que el usuario escoja y realiza la conexión con la base de datos correspondiente, generando la respuesta apropiada en caso que la conexión no se pueda realizar, o dirigiendo la interfaz de usuario a la página de ingreso de sentencias SQL en caso contrario.

La vista fue implementada utilizando la tecnología JSP de Java, y además con páginas HTML estándar, JavaScript y hojas de estilo (CSS) para dar un apariencia distinta al que por defecto define el sistema operativo.

Cada página JSP muestra información al usuario para que este interactúe con el sistema o para mostrar información proveniente de un evento que el mismo generó. Por ejemplo, la pantalla de entrada al sistema:

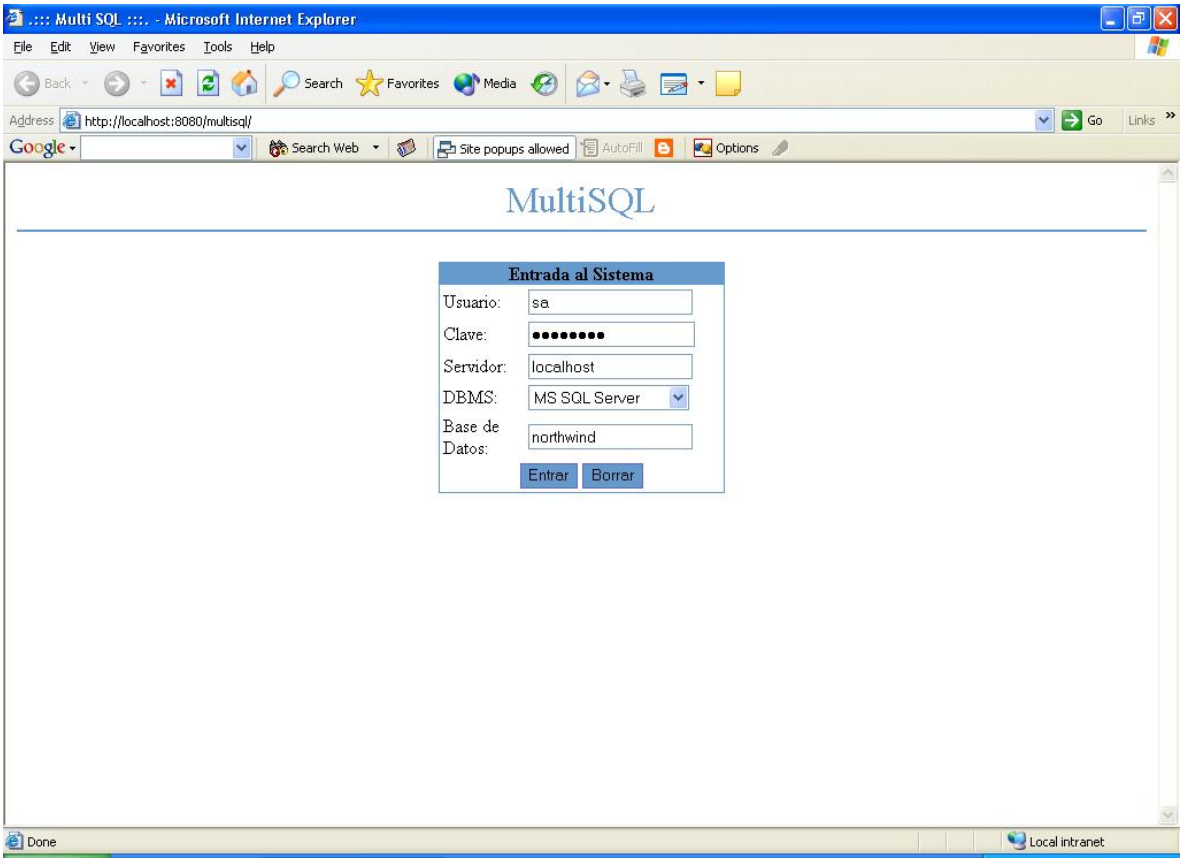

Figura 46. Entrada el Sistema

La dirección desde donde se puede acceder a la aplicación es:
http://multisql.unitecnologica.edu.co:8080/multisql/

Para ingresar como administrador, la dirección es:
http://multisql.unitecnologica.edu.co:8080/multisql/admin.jsp

El usuario es: admin y la clave inicial del administrador es: admin

Esta página le presenta al usuario los parámetros que debe ingresar para poder realizar una entrada exitosa al sistema. Fue implementada con JSP, como se muestra a continuación:

```
<%@page language="java" errorPage="error.jsp" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>.::: Multi SQL :::.</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<link rel="stylesheet" type="text/css" href="styles.css"/>
</head>
<body>
<%@include file="header.html" %>
<form name="entrada" action="MainServlet" method="POST">
<input type="hidden" name="cmd" value="LOGIN"/>
<input type="hidden" name="redirect" value="SQL"/>
<table class="conborde" align="center" cellspacing="0" cellpadding="0">
<tr>
<th class="relleno" colspan="2">Entrada al Sistema</th>
</tr>
<tr>
<td>
        <table border="0">
        <tr>
        <td width="30%">Usuario:</td>
        <td width="70%"><input type="text" name="usuario" size="20" value="sa"/></td>
        </tr>
        <tr>
        <td width="30%">Clave:</td>
        <td width="70%"><input type="password" name="clave" size="20"
value="45692901"/></td>
        </tr>
        <tr>
        <td width="30%">Servidor:</td>
        <td width="70%"><input type="text" name="host" size="20"
```

```
value="localhost"/></td>
        </tr>
        <tr>
        <td width="30%">DBMS:</td>
        <td width="70%">
                <select name="dbms">
                        <option value="seleccionar">Seleccionar...</option>
                        <option value="ORACLE">Oracle</option>
                        <option value="INTERBASE">Interbase (Firebird)</option>
                        <option value="MYSQL">MySQL</option>
                        <option selected value="MSSQL">MS SQL Server</option>
                </select>
        </td>
        </tr>
        <tr>
        <td width="30%">Base de Datos:</td>
        <td width="70%"><input type="text" name="db" size="20" value="northwind"/></td>
        </tr>
        <tr>
        <td align="center" colspan="2">
        <input type="submit" name="enviar" value="Entrar" class="button"/>
        <input type="reset" name="borrar" value="Borrar" class="button"/>
        </td>
        </tr>
        </table>
</td>
</tr>
</table>
<% if (request.getAttribute("msg") != null) { %>
<center><font face="Times New Roman" size="+1" class="alerta"><%=
request.getAttribute("msg") %></font></center>
<% } %>
</form>
```

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

Se puede ver en esta implementación la manera como se mezcla código HTML estándar con código Java, característica de la tecnología JSP, que permite dar dinamismo a la aplicación. Con el código Java insertado se recupera la información que envía el modelo y se visualiza en la pantalla.

Las aplicaciones Web basadas en Java, deben seguir un estándar en la manera como se organizan los archivos y todos los recursos de la aplicación para que puedan ser publicados en el servidor de aplicaciones. Todas las aplicaciones Web deben entonces seguir el siguiente esquema de directorios cuando están publicadas en el servidor:

Directorio raiz (/)

- Documentación (/docs)
- Descriptor de la aplicación (/WEB-INF/web.xml)
- Servlets, JavaBeans, clases Helper y archivos de configuración utilizados por estas clases (/WEB-INF/classes)
- Bibliotecas de funciones, Drivers JDBC (/WEB-INF/lib)

Para una mejor organización del código fuente de la aplicación, se utilizó la siguiente estructura de directorios durante la etapa de desarrollo:

Directorio raiz (/)

- Documentación (/docs)
- Fuentes de Java (/src)
- Contenido estático, JSP, HTML (/web)
- Descriptor de la aplicación (/web/WEB-INF/web.xml)
- Directorio de código compilado (/web/WEB-INF/classes)

En el directorio raiz se encuentran todos los archivos de propiedades (*.properties) usados por la aplicación, así como el archivo de tareas de Apache Ant, programa que

permite el deployment automatizado de la aplicación sobre el servidor de aplicaciones Apache Tomcat.

En el directorio /docs se colocan los archivos README de la aplicación y la documentación HTML generada por Javadoc.

En el directorio /web se encuentran los archivos .jsp, las páginas estáticas HTML, las hojas de estilo .css, los scripts de JavaScript y, en general, todo lo que queda en el directorio raiz de la aplicación al publicarse en el servidor Tomcat, es decir, todo lo que es visible al usuario a través del URL.

El descriptor de la aplicación (web.xml) es un archivo en lenguaje XML que, como su nombre lo indica, describe la aplicación. En el se especifican parámetros de inicialización para la aplicación, los servlets que la componen, el mapeo que se hace de dichos servlets al momento de construir el URL para su llamado, y cualquier parámetro inicial que se quiera pasar a los mismos.

El directorio /web/WEB-INF/classes es generado automáticamente por Apache Ant al momento de ejecutar la tarea de compilación de código fuente y guarda en el las clases de los JavaBeans, los componentes de acción, y demás código Java necesario para la aplicación, como el Parser generado por ANTLR.

Apache Ant, es una herramienta para automatizar el proceso de desarrollo y publicación de una aplicación. Está implementada usando el lenguaje Java y es un proyecto que nació de la necesidad que tenían los desarrolladores de reemplazar al obsoleto “make” por una herramienta más eficiente y, además, multiplataforma.

Ant recibe como entrada un archivo de configuración (build.xml por defecto), el cual contiene las reglas y parámetros necesarios para realizar el proceso de desarrollo de la aplicación.

En los anexos podrá observar el archivo build.xml utilizado para el desarrollo de esta aplicación.

8. APORTES Y VENTAJAS DE LA APLICACION

8.1 APORTES Y VENTAJAS.

Primero que todo, se quiere recalcar que el objetivo primario de la aplicación es: servir de herramienta única para el aprendizaje del lenguaje SQL, independientemente del motor de bases de datos que se utilice para hacer las consultas; esta independencia nos garantiza que el estudiante se puede centrar en el estudio del lenguaje SQL y no en las características de la herramienta de conexión propias del motor de bases de datos (SQL Plus en el caso de Oracle o Query Analizer en el caso de MS-SQL Server). Lógicamente, las herramientas propietarias incluyen características mucho mas poderosas y variadas, pero el objetivo no es aprender a manejar herramientas propietarias (cosa que se puede aprender con el estudio de los manuales una vez ya se cuenta con la fundamentación de la teoría de base de datos y del lenguaje SQL), sino aprender las bases del lenguaje SQL desde una herramienta gratuita, disponible para instalarla en cualquier servidor web; con tutores en línea que pueden fomentar aun mas el aprendizaje on line y autónomo y de esta manera disminuir la curva de tiempos de aprendizaje de bases de datos en asignaturas como bases de datos y/o los seminarios propios que se dictan orientados a cada motor.

Como ventaja adicional, la herramienta también permite que se use un lenguaje particular no estándar con el motor de base de datos que se desee. Esto se logra enviando todo a ejecutar en el servidor sin usar la revisión local (parser local).

Adicionalmente el aporte más importante de esta herramienta para el ambiente académico es que es una herramienta gratuita, liviana y disponible para descargar e instalar en su Intranet o en el web, lo que permite que el estudiante no tenga que tener un cliente de base de datos en su equipo para conocer las ventajas de la misma y obtener sus beneficios.

Las salas y horarios disponibles de las salas generalmente son limitadas y es muy beneficioso si el estudiante cuenta con una herramienta para realizar prácticas de SQL y un tutor en línea que puede visualizar desde la misma herramienta. Si se hace la instalación de la herramienta en un servidor web local a la institución que imparte la enseñanza para conectarse al DMBS con que se cuenta, el estudiante podrá tener acceso al curso on line, desde su hogar u otro sitio donde tenga acceso al Web sin las limitaciones espaciales generadas por los horarios y cantidad de equipo disponibles en muchas de las salas informáticas de nuestras universidades y/o otras instituciones Latino-Americanas de enseñanza en éstas áreas.

Los clientes de motores de bases de datos son generalmente costosos, para algunas instituciones de presupuestos limitados donde lo que interesa es el aprendizaje del SQL y/o la interacción con los DBMS en sus características más generales. Este proyecto pretende aportar al ámbito académico una herramienta básica para enseñanza del SQL estándar y las características nativas de los motores de bases de datos más accesibles en el mercado. El producto es una aplicación cliente/servidor que funcionara con interfaz web (en la intranet o desde internet), realizada en Java, totalmente orientada a Objetos, que permitirá el acceso a una de múltiples bases de datos con total independencia del motor. Lo cual no hubiera sido posible hacerlo tan fácilmente con PHP por ejemplo.

Lo interesante de este proyecto fue proveer una interfaz única y gratuita realizada en Java, que incorpora conceptos avanzados de compiladores (Generadores de parsers), con metodologías computacionales modernas para el diseño de aplicaciones para el Web; combinado tecnologías y lenguajes de ultima generación (Programación orientada a Objetos, Java, JSP; Servelts, JavaBeans, Arquitectura Cliente-Servidor Multicapa), para generar código optimo, liviano, independiente del DBMS, que funciona sobre internet y/o en la intranet y que permitirá al estudiante o al tutor del mismo obtener los siguientes beneficios:

- Interfaz amigable y estándar para tratamiento de instrucciones SQL operada desde el web.

- Selección de una de múltiples bases de datos para conexión (Oracle, MySQL, Postgresql, Interbase (FireBird), MSSQL Server, inicialmente)

- Parseo local del subconjunto de sentencias DML del lenguaje de consultas SQL(ANSI92) para bajar la carga del Servidor DBMS

- Parseo remoto para Compilación de instrucciones nativas a la base de datos que no hacen parte del ANSI SQL92, y se ejecutarán directamente sobre el servidor de base de datos.

- Fácil administración e instalación de la aplicación sobre un servidor de aplicación propio.

- Facilidades para el mantenimiento de la aplicación, teniendo como base un servidor de aplicación Apache TomCat. Version 4.x o superior.

- Facilidades para la ampliación del código (Mantenimiento) por las características del modelo Orientado a Objetos, y la utilización del paradigma de programación MVC (MODEL_VIEW_CONTROLLER)

- Ambiente cliente/Servidor multicapa

- Uso de Gramática BNF para generación del parser de manera automática.

- Uso de generadores de parser con código Java a partir de la gramática

- Estudiar on-line un tutorial Web de SQL estándar desde la misma aplicación

- Aplicar el lenguaje nativo del motor con que se cuente.

El problema se resolvió diseñando una herramienta con una interfaz única a una de múltiples bases de datos, orientado a objetos, totalmente gratuita basada en Web que por

su naturaleza abre a la comunidad académica otros espacios para gestionar su aprendizaje del Lenguaje de Bases de datos mas utilizado mundialmente (SQL).

Esperamos que la herramienta sea utilizada en otras instancias que no solo sea nuestra universidad en Cartagena, sino que se divulgue su uso en otras instituciones de bajos recursos en nuestra ciudad. Actualmente se esta contactando a las instituciones de mas bajos recursos para compartir los logros obtenidos.

8.2 COMPARACIÓN CON OTRAS HERRAMIENTAS

Existen en el mercado herramientas gratuitas y en su mayoría comerciales, algunas mucho mas poderosas desde el punto de vista de las opciones de uso de las mismas. Por ejemplo pueden poseer funciones avanzadas como son mantenimiento y admón. de versiones de software(scripts); lo cual es muy importante para herramientas cuya orientación de uso sea comercial o empresarial. Pero la mayoría de ellas, para el área de nuestro interés (docencia) en ambiente Web y multiplataforma tienen una o más de las siguientes desventajas:

- Permiten conectividad con un solo motor de BD exclusivamente.
- No son desarrolladas con el poder de la portabilidad que obtenemos con Java.
- Funcionan con ODBC, u OLEDB y no JDBC,
- No están orientadas al web (solo funcionan como clientes windows, por ejemplo).

A continuación mostramos algunas de sus características mas importantes y al final un cuadro que resume sus Ventajas y/o desventajas comparativamente con nuestro desarrollo.

CuteSQL 1.1

Es una herramienta de gran utilidad para todos aquellos que trabajen habitualmente con bases de datos, sobre todo si utilizan distintos sistemas comerciales. Permite ejecutar comandos y realizar peticiones SQL contra cualquier base de datos, con soporte para Oracle, MS SQL, Interbase y DB2. Permite conexiones nativas, vía ODBC o vía ADO pero no JDBC.

Incorpora una interfaz de uso sencillo, que muestra las peticiones SQL clasificadas, guarda un historial de los scripts utilizados y permite cambiar ciertos parámetros de configuración.[19]

SQLConnect 1.4.1

SQLConnect es una aplicación Java que te permite trabajar con SQL y ejecutar cualquier query (consulta o petición) a la base de datos, modificar o editar los datos contenidos en ella y en general cualquier tipo de manipulación.[20]

Este programa ofrece diversas características que facilitarán enormemente el trabajo con SQL, y que hacen de SQLConnect una completa y potente herramienta para administrar bases de datos. Entre las principales particularidades de SQLConnect encontrarás:

- Editar los datos a través de un formulario
- Exportar los resultados a un fichero ASCII
- Crear nuevas tablas o editar tablas de datos
- Navegar por la base de datos a través de un cómodo interfaz

SymphonyX 1.0

SymphonyX es una herramienta de fácil uso y con unas altas prestaciones que permite realizar consultas a través de ODBC y a bases de datos Access.[21]

- Algunas de sus principales opciones son:
- Multi-conexión
- Exportación de registros a Excel
- Impresión de registros
- Configuración de sentencias SQL
- Totalmente configurable

[19] http://cajamadridc.softonic.com/informacion_extendida.phtml?n_id=18870&plat=1
[20] http://programas.musikeiro.com.ar/index.php?seccion=verD&id_descarga=6123
[21] http://ozu.softonic.com/ie/13636

Es una utilidad, construida en Visual Basic 6.0 usando ADO, creada principalmente para ayudar al desarrollador. Consta de una interfaz muy intuitiva con la capacidad de los ya mencionados skins, los cuales ayudan aún más en la faceta de la facilidad de uso.

ifxSQL 1.0.2002.09.19

La interfaz es un entorno de múltiples ventanas en el que podremos realizar cuatro tipos de tareas:

- Recorrer la estructura de la base de datos con la ventana de Objetos
- Realizar selects a la base de datos con la ventana de Consulta
- Ejecutar sentencias de modificación de la base de datos, de alteraciones de tablas, creación de procedimientos, etc., con la ventana de Script
- Ver, editar, añadir o borrar filas de una tabla con la ventana de Tabla
- También permite realizar informes o exportar los datos a otros programas.[22]

Easy Query 1.0.15

Easy Query está construido en ADO y realiza consultas a múltiples orígenes de datos ODBC y bases de datos Microsoft Access 97/2000, habiéndose probado con DB/2, SQL Server y Access.

Permite hacer rápidas consultas SQL, usando cursor adUseServer, a cualquier origen de datos (SQL Server , DB/2 y Oracle), exportar registros a Microsoft Excel, HTML, texto, CVS y XML, configurar las sentencias SQL, multi-transacción por conexión, etc.

Además de todo esto también puede abrir, guardar e imprimir sentencias SQL, e imprimir los registros.[23]

[22] http://ifxsql.softonic.com/ie/23228
[23] http://easy-query.softonic.com/ie/20667

WinSQL 3.2.8.440

WinSQL permite enviar sentencias SQL (mejor conocidas como "queries") y comandos a cualquier base de datos SQL. También puede usarse para chequear el catálogo de la base de datos. Permite enviar múltiples sentencias a múltiples fuentes de datos al mismo tiempo.

WinSQL también incluye un asistente para la publicación HTML que permite publicar datos en Web, así como un asistente de sentencias SQL que ayuda a crear sentencias SQL a aquellos que no estén muy familiarizados con el lenguaje SQL.[24]

TOAD 6.3.6.1 (Tool for Oracle Admin and Devdelopers)

Toad permite a los programadores en Oracle trabajar de forma mucho más cómoda e intuitiva en sus bases de datos, gracias a una interfaz totalmente gráfica que dispone de todas las herramientas que se requieran.[25]

Esta utilidad permite desarrollar aplicaciones con mayor rapidez y facilidad, simplificando las tareas de administración de bases de datos y con posibilidad de trabajar en varios documentos simultáneamente, incluso si son de distintos tipos como HTML, Java, SQL o texto.

Incorpora un editor avanzado que te permite el uso de combinaciones de teclas, plantillas, corrector integrado y autocompletado de comandos para una mayor comodidad a la hora de programar.

Un navegador interno permite además visualizar la base de datos y gestionar objetos con facilidad. El programa es capaz además de conectarse a varias bases de datos simultáneamente

En la tabla 9 se muestra un resumen del comparativo de las herramientas mencionadas.

[24] http://ask.softonic.com/ie/9591/WinSQL
[25] http://cajamadridc.softonic.com/informacion_extendida.phtml?n_id=26344&plat=1

Nombre	Tamaño	Funcionalidad	Desventajas	Licencia	Plataforma
CuteSQL 1.1	1.41MB	Ejecuta comandos y peticiones SQL a cualquier base de datos	Solo permite conexiones nativas, vía ODBC o vía ADO. Solo para ambientes Windows	Freeware	Win95/98/ME/ WinNT/2000
SQLConnect 1.4.1	183KB	Potente herramienta de trabajo con bases de datos SQL	la versión es shareware (Caduca a los 30 días de uso) Es comercial (costo licencia) Solo para ambientes windows	ShareWare	Win95/98/ME/ WinNT/2000
SymphonyX 1.0	3.71MB	Consulta bases de datos vía ODBC	Las consultas son solo vía ODBC No soporta JDBC ni OLEDB	Freeware	Win95/98/NT
ifxSQL 1.0.2002.09.19	1.04MB	Acceso nativo a bases de datos Informix	Solo diseñada para Informix	Freeware	Win95/98/ME/ WinNT/2000/XP
Easy Query 1.0.15	2.09MB	Consulta orígenes de datos ODBC y bases de datos Access	Solo para ambiente windows Solo para origenes de datos ODBC	Freeware	Win95/98/ME/ WinNT/2000/XP
WinSQL 3.2.8.440	726KB	Prueba sentencias SQL en cualquier BdD ODBC	Solo para ambiente windows Solo para orígenes de datos ODBC	*Freewar*	Win95/98/ WinNT/2000
TOAD 6.3.6.1 (Tool for Oracle Admin and Devdelopers)	2.33 MB	Consulta bases de datos Oracle	No permite escribir mas de una línea y editarla Conecta solo a bases de datos Oracle y no a múltiples Bases de datos.	ShareWare	Win95/98/ WinNT/2000

Tabla 9. Comparativo entre herramientas

9. CONCLUSIONES

Se ha logrado implementar una aplicación Web, con base en tecnologías de última generación como son Servlets y JSP, aplicando modelos de programación de vanguardia como MVC; esto se ha podido realizar gracias a la madurez de dichas tecnologías y a que encontramos disponibles en la comunidad de desarrolladores muchas herramientas que facilitaron la implementación en cada una de las fases del desarrollo: Diseño, Desarrollo y Deployment.

Cabe anotar que existen muchas otras metodologías de programación de aplicaciones Web en Java, como por ejemplo J2EE (Java 2 Enterprise Edition), dentro de cuya especificación se encuentran los Servlets y JSP, pero que incluyen otros modelos y otras tecnologías más complejas para desarrollo como lo son el uso de EJB (Enterprise JavaBeans). Para esta aplicación se utilizaron las tecnologías más sencillas pero que nos permitieron lograr los objetivos propuestos eficientemente y a bajo costo.

La principal ventaja de este desarrollo es proporcionar una herramienta que permita conectividad con múltiples manejadores de Bases de datos sin necesidad de incurrir en el uso de herramientas propietarias costosas., disponible para descargar e instalar en su servidor web local con todas las ventajas mencionados en la sección 8.1 y 8.2.

Las herramientas cliente de la mayoría de los DBMS actuales no son gratuitas y solo permiten la conectividad con su base de datos nativa. Muchas de las gratuitas tienen limitación por el sistema operativo que soportan o el lenguaje en que fueron desarrolladas no es portable y no está disponible para hacer mejoras.

La ventaja de este tipo de desarrollos es la independencia que se puede lograr de los proveedores actuales de herramientas de BD que crean monopolios de software con altos

costos de licenciamiento y la portabilidad que se puede implementar a diferentes ambientes operativos de forma paulatina.

La aplicación se entrega en un estado de completa funcionalidad, a nivel de ejecución de sentencias sobre los distintos motores que soporta, tiene la capacidad de agregar motores de bases de datos adicionales, y verifica la sintaxis de sentencias DML directamente con el DBMS nativo. Sería útil pensar en complementar la aplicación con compiladores específicos para cada tipo de motor de bases de datos con el que la aplicación se conecta y que se carguen dinámicamente dependiendo de la escogencia del usuario, esto para lograr minimizar la carga en los servidores y poder verificar sintaxis más específicas de sentencias DDL directamente en el cliente. Esta es una labor interesante pues además de requerir recopilar todas las sintaxis específicas de DDL para los distintos motores implica implementarles el verificador de sintaxis de manera independiente, pero no es una tarea imposible de realizar. Existen en Internet muchos sitios en los que encontramos las definiciones gramaticales en formato BNF para el lenguaje DDL de cada motor, lo cual es la materia prima para la construcción del compilador.

Esperamos que muchas mejoras sean incorporadas al código a través de la página donde estará disponible el código y seguir enriqueciendo este desarrollo con mas tutoriales en línea e hipervínculos a otros tutoriales.