Universidad de las Ciencias Informáticas
FACULTAD 6

Título: Intérprete SQL para el módulo Diseñador de Consultas del GDR

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor: Jorge Antonio Bernal Rojas

Tutor: Ing. Vania Elena Yanes León

La Habana, mayo de 2013
"Año 55 de la Revolución"

"Lo que caracteriza al hombre de ciencia no es la posesión del conocimiento o de verdades irrefutables, sino la investigación desinteresada e incesante de la verdad."

KARL POPPER (1902 - 1994)

Declaración de Autoría

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Jorge A. Bernal Rojas

_____

Firma del Autor

Vania E. Yanes León

_____

Firma del Tutor

Datos de Contacto

Tutor: Vania Elena Yanes León
Universidad de las Ciencias Informáticas, La Habana, Cuba.
Correo electrónico: veyanes@uci.cu
La tutora Vania Elena Yanes León es graduada de Ingeniero en Ciencias Informáticas actualmente forma parte del departamento de Integración de Soluciones del centro DATEC donde ocupa el cargo de Analista del proyecto GDR y Asesora de Investigación y postgrado del departamento Integración de Soluciones.

Autor: Jorge Antonio Bernal Rojas
Universidad de las Ciencias Informáticas, La Habana, Cuba.
Correo electrónico: jabernal@estudiantes.uci.cu

DEDICATORIA

Dedico especialmente este trabajo a mi abuelo Francisco Rojas y a mi Tío Emiliano López que dejaron de estar presente entre nosotros hace pocos años.

A mis abuelas Leyda López y Teresa Vicente por el amor y los años de su vida que dedicaron a mantener unida a esta bonita familia, espero poder tenerlas muchos años más.

A mi mamá Teresa Rojas y a mi papá Jorge Luis Bernal porque sin su amor, constante apoyo y confianza no hubiese llegado jamás hasta aquí, por ser ambos un digno ejemplo a seguir.

A mis Tías-mami Leyda Jaime y Nancy Guzman por su apoyo incondicional y haberse convertido ambas en mi segunda madre, gracias por quererme como un hijo.

A mi hermano, Luis Eduardo Bernal por quererme mucho y ser un gran amigo para mi, que sirva mi esfuerzo de motivación.

A mi familia en general por esperar siempre lo mejor de mí.

AGRADECIMIENTOS

Agradezco a mis padres y a mi Tía Leydita, pues ellos junto a mi han realizado los mayores sacrificios, sin su ayuda nada de esto hubiese sido posible, no existen palabras para agradecerles, ni para decirles lo mucho que los quiero.

A mis abuelas Leyda López y Teresa Vicente por sus consejos y por el amor que me han brindado en toda mi vida.

A mi familia por todo el cariño y la comprensión demostrada todo este tiempo, en especial a mi tía Nancy por acogerme como un hijo.

A mi novia Yuned Rivero por apoyarme en todo este tiempo que hemos estado juntos, por ser mi leal compañera en los momentos más difíciles y por ser la solución en muchos de estos momentos, por quererme y comprenderme, por todos los momentos felices que hemos pasado juntos.

A mi tutora Vania Elena Yanes León por haberme ayudado, por la confianza que depositó en mí desde el primer momento, por su tiempo y dedicación.

A mis compañeros de grupo y de apartamento que hemos navegado en el mismo barco y aunque algunos quedaron en el camino la mayoría llegamos a tierra firme.

A mis amigos Fernando, Jeovany, Ernesto, Darlon, Miguel Verdecia, Miguel Cordero, Reisel Betancourt, Reisel Gonzales, Eudel, Raul, Liandry, Justin, Yailema, Sureny, Grey, Yanet, Yosbel por todos los momentos buenos, malos, de intenso estrés y mucha diversión que compartimos en todos estos años, nunca los olvidaré.

A mi tribunal y a mi oponente, por corregir cada detalle y estar dispuestos a ayudarme.

Agradecimientos

A mis profesores todos, aún a los que ya no puedo recordar, puesto que todos han contribuido a mi formación como profesional y como persona.

A Fidel Castro y a la Revolución por darme la posibilidad de convertirme en un profesional y a la Universidad de las Ciencias Informáticas por acogerme durante estos 5 años.

RESUMEN

El departamento de Integración de Soluciones del Centro de Tecnologías y Gestión de Datos (DATEC), se dedica al desarrollo de sistemas informáticos encaminados a satisfacer las necesidades de gestión de la información destacando la utilización del lenguaje SQL para la interacción con bases de datos. Entre estos sistemas se encuentra el Generador Dinámico de Reportes (GDR), que cuenta con el módulo Diseñador de Consulta. Este funciona como un editor gráfico de consultas SQL permitiendo crear, editar y ejecutar consultas del tipo antes mencionado y consta de dos vistas básicas: la vista de diseño y la vista SQL. La vista de diseño es donde se realiza el diseño gráfico de las consultas y la vista SQL es donde se muestra el código fuente perteneciente a la consulta en diseño. La versión actual del GDR permite actualizar automáticamente la vista SQL una vez que se haya realizado cambios en la vista de diseño, sin embargo no ocurre así de forma contraria, este problema afecta considerablemente la usabilidad del módulo. El presente trabajo de diploma tiene como objetivo desarrollar un intérprete SQL que contribuya a la creación de reportes a partir de consultas prediseñadas utilizando el módulo Diseñador de Consultas. El resultado fundamental se centra en la obtención de un intérprete SQL para el módulo Diseñador de Consulta del GDR potenciando en gran medida la incorporación de nuevas funcionalidades contribuyendo de manera positiva al diseño de reportes de forma dinámica e intuitiva.

Palabras clave: consulta SQL, Diseñador de Consulta, intérprete.

Tabla de contenido

Índice de Tablas

Índice de Figuras

INTRODUCCIÓN

En el mundo actual, la información ocupa un lugar importante estando presente en todos los niveles y ramas de la economía, la política y la sociedad, teniendo en cuenta que el análisis realizado a partir de ella constituye la base para la toma de decisiones, de ahí que es vital para emprender proyectos en cualquier esfera. A lo largo de los años la cantidad de información se va haciendo cada vez más amplia y con la aparición de las nuevas tecnologías ha evolucionado la forma en que se desarrollan los sistemas de información de las organizaciones.

El almacenamiento de los datos procesados por estos sistemas es llevado a cabo a través de bases de datos debido a la gran necesidad de organizar y gestionar grandes volúmenes de información. Estas representan el núcleo de dichos sistemas y teniendo en cuenta varios criterios pueden ser clasificadas de acuerdo a sus características y propósitos, al igual que existen diferentes modelos. Un modelo de datos es básicamente una descripción del contenedor de la información, así como de los métodos para almacenar y recuperar información de esos contenedores. Entre los modelos que con frecuencia se utilizan para el diseño de las bases de datos se destaca el modelo relacional, por ser el más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente debido a su principal característica desde el punto de vista estructural de organizar la información en forma de tablas. Dicha forma de almacenar información presenta un lenguaje común de acceso a los datos, facilitando la programación de aplicaciones con bases de datos, consiguiendo que los programas sean independientes de los aspectos físicos de las mismas.

A este lenguaje se le denomina: Lenguaje de Consulta Estructurado SQL (por sus siglas en inglés Structured Query Language) creado en los laboratorios de investigación de la empresa multinacional estadounidense IBM[1], usado para recuperar y manipular datos en una base de datos relacional. Este lenguaje permite efectuar consultas con el fin de manejar de forma sencilla información de interés contenida en bases de datos relacionales, así como modificarlas, a este proceso básicamente se le

[1]IBM (por sus siglas en inglés International Business Machines) es una empresa multinacional estadounidense de tecnología y consultoría que fabrica y comercializa hardware y software para computadoras, ofrece servicios de infraestructura, alojamiento de Internet, y consultoría en una amplia gama de áreas relacionadas con la informática.

denomina consulta SQL la cual ofrece una amplia flexibilidad y poder para administrar datos. La creciente aceptación de SQL es una de las tendencias importantes en la industria informática hoy en día. Durante los últimos años se ha convertido en el lenguaje de base de datos estándar, alcanzando un lugar sobresaliente como tecnología informática importante y como fuerza de mercado poderosa.

El departamento de Integración de Soluciones del Centro de Tecnologías y Gestión de Datos (DATEC), perteneciente a la infraestructura productiva de la Universidad de las Ciencias Informáticas (UCI) se dedica al desarrollo de sistemas informáticos encaminados a satisfacer las necesidades de gestión de la información, destacando al lenguaje SQL en su utilización para la interacción con bases de datos. Entre estos sistemas se encuentra el Generador Dinámico de Reportes (GDR), este sistema permite obtener diferentes reportes en los sistemas de gestión de la información con el objetivo de tomar decisiones, realizar estudios investigativos, o sencillamente consultar información propia del negocio en el cual esté siendo utilizado. El GDR es una herramienta web que permite a sus clientes consultar los gestores de bases de datos de sus organizaciones y poder generar reportes con la información que estos manejan. Consta de 6 módulos: Diseñador de Modelos, Diseñador de Reportes, Diseñador de Consultas, Visor de Reportes, Administrador de Reportes y Seguridad. El módulo Diseñador de Consulta funciona como un editor gráfico de consultas SQL permitiendo crear, editar y ejecutar consultas del tipo antes mencionado y consta con dos vistas básicas: la vista de diseño y la vista SQL. La vista de diseño es donde se realiza el diseño gráfico de las consultas y la vista SQL es donde se muestra el código fuente perteneciente a la consulta en diseño, y además permite realizar cambios a dicho código. La versión actual de GDR permite actualizar automáticamente la vista SQL una vez que se haya realizado cambios en la vista de diseño, sin embargo no ocurre así de forma contraria, si se realizan cambios en el código de la vista SQL, la vista de diseño no es capaz de actualizar su contenido automáticamente, este problema afecta considerablemente la usabilidad del módulo, debido que al ser utilizado por un usuario con conocimientos en el lenguaje SQL capaz de modificar el código de una consulta en diseño, esta no se graficará correctamente.

En función de lo antes descrito, en el presente trabajo se plantea como **problema de la investigación**: ¿Cómo contribuir a la creación de reportes a partir de consultas prediseñadas utilizando el módulo Diseñador de Consultas del Generador Dinámico de Reportes?

Introducción

Se propone como **objeto de estudio**, traductores de lenguajes de programación de alto nivel, enmarcado en el **campo de acción** intérprete SQL para el módulo Diseñador de Consultas del Generador Dinámico de Reportes.

Para darle solución al problema planteado se traza como **objetivo general**: desarrollar un Intérprete SQL que contribuya a la creación de reportes a partir de consultas prediseñadas utilizando el módulo Diseñador de Consultas del Generador Dinámico de Reportes. Como **objetivos específicos** se plantea:

- Realizar el análisis y diseño del Intérprete SQL para el módulo Diseñador de Consulta del Generador Dinámico de Reportes.
- Implementar el Intérprete SQL para el módulo Diseñador de Consultas del Generador Dinámico de Reportes.
- Realizar pruebas que validen el correcto funcionamiento del Intérprete SQL para el módulo Diseñador de Consultas del Generador Dinámico de Reportes.

Para dar cumplimiento a los objetivos se precisan las siguientes **tareas** de la investigación:

- Análisis de los conceptos básicos y técnicos implicados en el desarrollo de un Intérprete SQL.
- Selección de las herramientas y metodología a utilizar para el desarrollo del Intérprete SQL.
- Identificación de los requisitos funcionales y no funcionales del Intérprete SQL para el módulo Diseñador de Consultas del Generador Dinámico de Reportes.
- Confección del modelo de casos de uso del sistema del Intérprete SQL.
- Elaboración del modelo de diseño del Intérprete SQL para el módulo Diseñador de Consultas del Generador Dinámico de Reportes.
- Elaboración del modelo de implementación del Intérprete SQL para el módulo Diseñador de Consultas del Generador Dinámico de Reportes.
- Desarrollo de la etapa de análisis del Intérprete SQL para el módulo Diseñador de Consultas del Generador Dinámico de Reportes.
- Desarrollo de la etapa de síntesis del Intérprete SQL para el módulo Diseñador de Consultas del Generador Dinámico de Reportes.
- Integración del Intérprete SQL al módulo Diseñador de Consultas del Generador Dinámico de Reportes.

- Realización de pruebas funcionales para comprobar el correcto funcionamiento del Intérprete SQL.
- Diseño de los casos de prueba a partir de los requisitos funcionales para llevar a cabo la realización de las pruebas.
- Documentación y corrección de las no conformidades identificadas en la ejecución de las pruebas.

El presente trabajo de diploma está estructurado en tres capítulos:

Capítulo 1. Fundamentación Teórica: en el capítulo se plantean los principales conceptos relacionados con los traductores de lenguajes de programación de alto nivel, se realiza un estudio detallado sobre las etapas de desarrollo de un intérprete así como sus fases. Se selecciona la metodología de desarrollo a utilizar y se exponen tecnologías y herramientas en las que se apoya la solución.

Capítulo 2. Análisis y Diseño del sistema: en el capítulo se describe el proceso llevado a cabo durante la fase de análisis y diseño. Se definen los requisitos funcionales y no funcionales del software lo cual es un paso de gran importancia, para su éxito. Se muestra la estructura del sistema a desarrollar a través de una representación visual de las clases conceptuales en el modelo de dominio. Se especifica la estructura física de la solución que se propone mediante el diagrama de casos de uso del sistema y el modelo de despliegue. Se muestran los diagramas de clases del diseño, por medio de los cuales se representa la estructura estática del sistema.

Capítulo 3. Implementación y Prueba: en el capítulo se define la gramática que guiará el proceso de interpretación y se realiza la especificación del lenguaje a interpretar. Se muestra el modelo de implementación que pone en práctica el diseño de la solución que se va a desarrollar, así como el diagrama de componentes del Intérprete SQL. Se procede a validar el funcionamiento del sistema con la aplicación de pruebas funcionales y de integración.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se platean los principales conceptos relacionados con el objeto de estudio que son de vital importancia para entender posteriormente la solución propuesta. Se plasma un estudio detallado sobre las etapas de desarrollo de un intérprete así como sus fases. Se seleccionan las herramientas, metodología y lenguajes a utilizar para el desarrollo de la solución.

1.1 Traductores de los lenguajes de programación de alto nivel

Un traductor es un programa que toma como entrada un programa escrito en un lenguaje de programación (lenguaje fuente) y produce como salida un programa en otro lenguaje (lenguaje objeto)[2]. Existen dos tipos principales de traductores de los lenguajes de programación de alto nivel: compilador e intérprete.

Compilador: analiza el programa fuente y lo traduce a otro equivalente escrito en otro lenguaje (por ejemplo, en el lenguaje máquina[3]). Su acción equivale a la de un traductor humano, que toma un libro y produce otro equivalente escrito en otra lengua. (1)

Intérprete: analiza el programa fuente y lo ejecuta directamente, sin generar ningún código equivalente. Su acción semeja a la de un intérprete humano, que traduce las frases que escucha sobre la marcha, sin producir ningún escrito permanente. (1)

Intérpretes y compiladores tienen diversas ventajas e inconvenientes que los hacen complementarios. Un intérprete facilita la búsqueda de errores, pues la ejecución de un programa puede interrumpirse en cualquier momento para estudiar el entorno (valores de las variables). Sobre la marcha el programa puede modificarse sin necesidad de volver a comenzar la ejecución. En cambio, un compilador suele generar programas más rápidos y eficientes, pues el análisis del lenguaje fuente se hace una sola vez durante la

[2] **Lenguaje objeto:** código generado por un compilador o un ensamblador, traducido a partir del código fuente de un programa. Casi siempre este término se refiere al código máquina que puede ejecutarse directamente en la unidad central de proceso (CPU) del sistema.

[3] **Lenguaje máquina:** lenguaje propio del ordenador, basado en el sistema binario, o código máquina, resulta difícil de utilizar para las personas.

generación del programa equivalente. Sin embargo el que más se ajusta a la posible solución del problema planteado es el intérprete, de acuerdo a las características antes mencionadas y las necesidades existentes que presenta el módulo Diseñador de Consulta del GDR.

1.2 Intérprete

En ciencias de la computación intérprete o interpretador es un programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Es un traductor que realiza la operación de compilación paso a paso. Para cada sentencia que compone el texto de entrada, se realiza una traducción, ejecuta dicha sentencia y vuelve a iniciar el proceso con la sentencia que le sigue. (2)

Se puede modelar el proceso de interpretación como el resultado de dos etapas. En la primera etapa se analiza la entrada para averiguar qué es lo que se intenta comunicar, esto se conoce como **análisis**. El fruto de esta etapa es una representación de la entrada que permite que la siguiente etapa se desarrolle con facilidad. La segunda etapa, la **síntesis**, toma la representación obtenida en el análisis y la transforma para obtener los resultados deseados.

Análisis:

El objetivo de esta etapa es obtener una representación de la entrada que nos permita realizar la síntesis o la interpretación con comodidad. Esta etapa se divide en tres fases fundamentales: análisis léxico, análisis sintáctico y análisis semántico. A continuación se explica en qué consiste cada fase de esta etapa de análisis.

- **Análisis léxico**

Esta fase se encarga de la división de la entrada en componentes léxicos, se analiza la entrada caracter a caracter y se agrupan en entidades sintácticas simples o elementales denominadas tokens. Cada uno de estos componentes se clasifica en una categoría y puede recibir uno o más atributos con información relevante para otras fases. El criterio que se emplea para clasificar cada componente es su pertenencia o no a un lenguaje.

Como el analizador léxico es el componente que lee el código fuente, realiza otras funciones colaterales, por ejemplo: eliminar los comentarios, caracteres tabs, espacios en blanco, caracteres de fin de línea y correlaciona los mensajes de error del intérprete con el código fuente conservando los números de líneas para luego asociarlos con mensajes de error. Cuando se habla de análisis léxico se utilizan tres términos

relacionados entre sí pero que tienen significados distintos: Token, Patrón y Lexema. A continuación aparecen plasmados los conceptos asociados a estos tres términos.

Token

Un token es un par formado por un nombre de símbolo y un atributo opcional valor. El nombre del símbolo es un símbolo abstracto que representa una especie de unidad léxica, por ejemplo, una palabra clave concreta, o una secuencia de caracteres de entrada que denota un identificador. Los tokens son los símbolos de entrada del analizador sintáctico. (3)

Para los tokens existen varios tipos de categorías sintácticas y para cada categoría sintáctica varios tipos de tokens asociados. Las categorías de tokens pueden variar de un lenguaje a otro, pero en general se distinguen las siguientes: (3)

- **Palabras reservadas:** cadenas predeterminadas por el lenguaje las cuales no se pueden usar como identificadores.
- **Identificadores:** son las variables del programa, los nombres de métodos y clases.
- **Constantes numéricas y literales:** los cuales a su vez podrán dividirse en tres: literales reales, literales enteras y literales de cadena.
- **Operadores:** los cuales a su vez podrán dividirse en 4 tipos: aritméticos, relacionales, lógicos, y de asignación.
- **Símbolos de puntuación:** aquí se incluyen los puntos, las comas, los paréntesis, las llaves y todos los símbolos de puntuación que pertenezcan a un lenguaje determinado. Pueden dividirse en varios tipos, todo depende del significado sintáctico que tenga un determinado símbolo de puntuación en el lenguaje.

Patrón

Un patrón es la descripción de la forma que pueden tomar los lexemas. En el caso de una palabra clave, el patrón es la secuencia de caracteres que forman la palabra clave. Para los identificadores y otros, el

patrón se puede representar con una expresión regular, aunque para la implementación es más conveniente representarlos usando un autómata finito[4]. (3)

Lexema

Un lexema es una secuencia de caracteres en el programa fuente que coincide con el patrón de un token y se identifica por el analizador léxico como una instancia de ese token. (3)

Para cada token identificado por el analizador léxico, es necesario almacenar cierta información: el lexema, el número de línea de su primera aparición y otros datos que dependen del tipo de token. Dicha información es necesaria para las fases posteriores del proceso de interpretación y por lo general para su almacenamiento se usa la tabla de símbolos.

Tabla de símbolos

Una tarea fundamental en un intérprete es la de almacenar los identificadores utilizados en un programa y sus atributos principales, de manera que en cualquier momento pueda conocerse de un identificador, su tipo y alcance. Esta información se almacena generalmente en una estructura conocida como tabla de símbolos la cual posee un conjunto de atributos que pueden variar de uno a otro lenguaje. Los tokens que representan constantes o identificadores se almacenan en la tabla de símbolos a medida que van apareciendo en el programa fuente. (3)

Durante todo el proceso de interpretación se invierte una parte significativa del tiempo en el manejo de la tabla de símbolos. En la fase de análisis léxico se insertan los símbolos según aparecen en el programa fuente y en las restantes fases se van agregando atributos a medida que se conocen. Para cada identificador que se analice es importante conocer sus atributos, por lo que el acceso a la tabla de símbolos se realiza constantemente. Aquí se hace evidente que las tablas deben organizarse de forma tal que permitan una búsqueda eficiente, es por eso que es muy importante seleccionar adecuadamente la estructura de datos que la represente. Existen múltiples formas de organizar las tablas de símbolos, en particular pueden tenerse tablas separadas para los diferentes tipos de tokens, es decir tablas para los

[4] Un autómata finito es una herramienta abstracta que se utiliza para reconocer un determinado lenguaje regular. Es un modelo matemático de un sistema que recibe una cadena constituida por caracteres de cierto alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce.

identificadores, tablas para las palabras reservadas, u organizar todos los tipos de tokens en una sola tabla general.

- **Análisis sintáctico**

El análisis sintáctico (parse) se encarga de encontrar las estructuras presentes en la entrada. A partir de lo que se recibe del analizador léxico, esta fase consiste en ir descubriendo las estructuras presentes en el código de acuerdo a una gramática definida y mediante estas estructuras identificadas construye un árbol sintáctico que permite expresar cómo se puede fabricar formalmente la entrada a partir de las reglas. Sin embargo, muchas veces el analizador sintáctico dirige el proceso de compilación, de manera que el resto de las fases evolucionan a medida que el parse va reconociendo la secuencia de entrada, por lo que, a menudo, el árbol ni siquiera se genera realmente.

En la práctica, el analizador sintáctico también realiza otro conjunto de acciones, las principales son:

- Controla el flujo de tokens reconocidos por parte del analizador léxico.
- Informa de la naturaleza de los errores sintácticos que encuentra e intenta recuperarse de ellos para continuar la compilación.

El tratamiento de los errores durante el proceso de análisis sintáctico debe cumplir al menos los siguientes requisitos:

1. Reportar la presencia de los errores de forma clara y precisa.
2. Recuperarse de los errores lo suficientemente rápido como para ser capaz de detectar los errores siguientes.
3. No demorar significativamente el procesamiento de los programas correctos.

El primer paso necesario para desarrollar el análisis sintáctico es definir la gramática del lenguaje y convertir la misma a una gramática de tipo LL1. A continuación se explica partiendo del concepto de gramática, cuándo estamos en presencia de una de tipo LL1.

Gramática

Una gramática es un conjunto de reglas para formar correctamente las frases de un lenguaje; así se tiene la gramática en el español. La idea para aplicar una gramática es que se parte de una variable, llamada símbolo inicial, y se aplican repetidamente las reglas gramaticales hasta que no existan variables en la

palabra. En ese momento se dice que la palabra resultante es generada por la gramática, o en forma equivalente, que la palabra resultante es parte del lenguaje de esa gramática. Tratándose de los compiladores se les llama “terminales” a los elementos del alfabeto Σ, y “no terminales” a las variables. (3)
Es posible restringir la forma de las reglas gramaticales de manera que se acomoden a patrones predeterminados. Por ejemplo, se puede imponer que el lado izquierdo de las reglas sea una variable, en vez de una cadena arbitraria de símbolos. Al restringir las reglas de la gramática se restringen también las palabras que se pueden generar.
Las gramáticas constituyen la mejor vía para la descripción sintáctica de los lenguajes de programación, brindando la posibilidad de añadir nuevas construcciones al lenguaje de forma fácil. Para lograr un análisis sintáctico eficiente y exitoso, la gramática definida por la cual este guía su proceso de análisis debe de cumplir ciertas condiciones:

- No recursiva a la izquierda.
- No ambigua.
- Factorizada por la izquierda.

Estas condiciones generalmente conducen a un tipo de gramática especial denominada LL1.

- **Análisis semántico**

Esta última fase es la que se encarga de comprobar que se cumplen las restricciones semánticas del lenguaje. El análisis semántico, toma como entrada el árbol sintáctico y comprueba si, además de las restricciones sintácticas, se cumplen otras restricciones impuestas por el lenguaje y que no pueden ser comprobadas mediante una gramática. Algunos ejemplos de estas restricciones son la necesidad de declarar las variables antes de usarlas, las reglas de tipos o la coincidencia entre los parámetros de las funciones en las definiciones y las llamadas. Como salida de esta fase, se obtiene una representación semántica: el árbol de sintaxis abstracta el cual sirve de entrada para la etapa de síntesis.

Síntesis:

Los intérpretes tienen como objetivo la obtención de los resultados del programa. Para ello deben realizar dos tareas: analizar su entrada y llevar a cabo las acciones especificadas por ella. En esta etapa se parte del árbol de sintaxis abstracta y se recorre junto con los datos de entrada para obtener los resultados.

A partir de la descripción de las etapas para el desarrollo de un intérprete realizado anteriormente, se puede representar la arquitectura de los intérpretes a través de la siguiente figura:

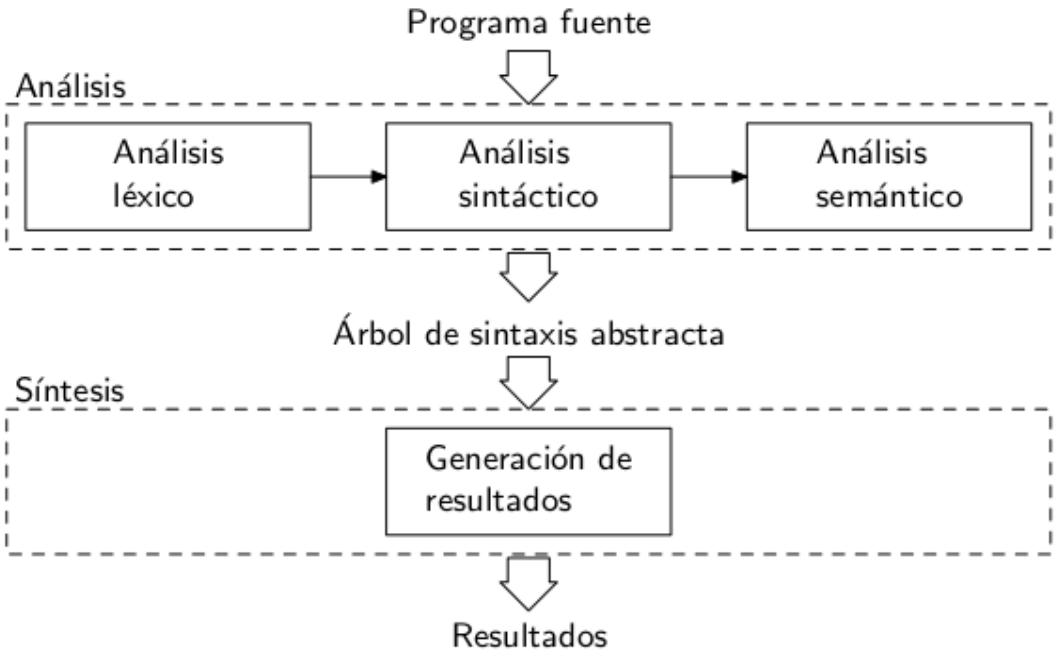

Figura 1 Arquitectura de los intérpretes. (4)

Después del análisis realizado sobre los intérpretes, es necesario puntualizar algunas características esenciales del lenguaje SQL debido a que el intérprete que se desarrollará está enmarcado en la interpretación de consultas SQL.

SQL

Structured Query Language (lenguaje de consulta estructurado), es un lenguaje para el acceso a bases de datos relacionales. Actualmente se ha convertido en un estándar de lenguaje de bases de datos, y la mayoría de los sistemas de bases de datos lo soportan, desde sistemas para ordenadores personales, hasta grandes ordenadores.(5)

Como su nombre indica, SQL permite realizar consultas a la base de datos y además realiza funciones de definición, control y gestión de la base de datos. Las sentencias SQL se clasifican según su finalidad dando origen a tres 'lenguajes' o sub-lenguajes:

El DDL (por sus siglas en inglés Data Definition Language), lenguaje de definición de datos, incluye órdenes para definir, modificar o borrar las tablas en las que se almacenan los datos y de las relaciones entre estas. Es el que más varía de un sistema a otro.

El DCL (por sus siglas en inglés Data Control Language), lenguaje de control de datos, contiene elementos útiles para trabajar en un entorno multiusuario. Entorno en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieran unos con otros.

El DML (por sus siglas en inglés Data Manipulation Language), lenguaje de manipulación de datos, permite recuperar los datos almacenados en la base de datos y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo nuevos datos, suprimiendo datos antiguos o modificando datos previamente almacenados. Las sentencias generadas por este lenguaje son las que serán interpretadas a través del intérprete SQL que se desarrollará posteriormente.

La mayoría de las sentencias SQL tienen la misma estructura. Todas comienzan por un verbo (select, insert, update, create), a continuación le sigue una o más cláusulas que indican los datos con los que se va a operar (from, where, group by, having, order by, limit), algunas de estas son opcionales y otras obligatorias como es el caso del from. SQL permite fundamentalmente dos modos de uso:

- Un uso **interactivo**, destinado principalmente a los usuarios finales avanzados u ocasionales, en el que las diversas sentencias SQL se escriben y ejecutan en línea de comandos, o un entorno semejante.

- Un uso **integrado**, destinado al uso por parte de los programadores dentro de programas escritos en cualquier lenguaje de programación anfitrión. En este caso SQL asume el papel de sublenguaje de datos.

En caso de hacer un uso embebido del lenguaje se pueden utilizar dos técnicas alternativas de programación. En una de ellas, el lenguaje se denomina SQL estático, las sentencias utilizadas no cambian durante la ejecución del programa. En la otra, el lenguaje recibe el nombre de SQL dinámico, se produce una modificación total o parcial de las sentencias en el transcurso de la ejecución del programa.
La utilización de SQL dinámico permite mayor flexibilidad y mayor complejidad en las sentencias, pero como contra punto obtenemos una eficiencia menor y el uso de técnicas de programación más complejas en el manejo de memoria y variables.

Características generales:
SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos. Es un lenguaje declarativo de alto nivel o de no procedimiento, que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación. De esta forma una sola sentencia puede equivaler a uno o más programas que utilicen un lenguaje de bajo nivel orientado a registro. (5)

1.3 Metodologías de Desarrollo de Software

Metodología de desarrollo en ingeniería de software es un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevos software. Estas garantizan la eficacia según los requisitos iniciales y la eficiencia al minimizar la pérdida de tiempo en el proceso de generación de software, indicando qué personas deben participar en el desarrollo de las actividades y qué papel deben tener. Este conjunto de procedimientos y técnicas detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla. Las metodologías de desarrollo tienen dos enfoques: enfoque tradicional/metodologías robustas y enfoque ágil/metodologías ágiles.

Las metodologías tradicionales o robustas imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Estas están centradas especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada. No se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno donde los requisitos no pueden predecirse o bien pueden variar. Entre ellas se destacan: RUP (Rational Unified Procces), MSF (Microsoft Solution Framework), Iconix, entre otras. (6)

Las metodologías ágiles combinan un conjunto de líneas de desarrollo donde resaltan la entrega sobre el análisis y el diseño, como principal objetivo estas buscan la satisfacción del cliente y la entrega temprana del software incremental. Son menos orientadas al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. Más orientadas al código, indicando que la parte más importante de la documentación es el código fuente. Entre ellas se destacan: XP (eXtreme Programming), Scrum, ASD (Adaptive Software Development) y OpenUP (Proceso Unificado Abierto). (6)

Las metodologías y técnicas existentes para el desarrollo de productos de software son analizadas y definidas en la arquitectura de cada proyecto. Para ello se realiza un estudio de algunas metodologías existentes y se realiza la selección de la que será empleada. A continuación se caracterizan las tres metodologías más utilizadas en la universidad:

RUP

RUP es una metodología robusta que define un proceso de desarrollo de software, esta precisa claramente quién, cómo, cuándo y qué debe hacerse en el proyecto. RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de reglas adaptables al contexto y necesidades de cada organización.

Características esenciales:

- Dirigido por los casos de uso: orientan el proyecto a la importancia para el usuario y lo que este quiere.
- Centrado en la arquitectura: relaciona la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden.

- Iterativo e incremental: dividiéndose el proyecto en mini proyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada.

Fases de desarrollo del software:

- Inicio: el objetivo en esta etapa es determinar la visión del proyecto. Se concreta la idea, la visión del producto y el alcance del proyecto.
- Elaboración: en esta etapa el objetivo es determinar la arquitectura óptima. Construir el producto, la arquitectura y los planes, hasta que el producto está listo para ser enviado a la comunidad de usuarios.
- Construcción: en esta etapa el objetivo es llevar a obtener la capacidad operacional inicial. Se basa en la elaboración de un producto totalmente operativo y en la elaboración del manual de usuario.
- Transición: el objetivo es llegar a obtener la liberación del proyecto. Se realiza la instalación del producto en el cliente y se procede al entrenamiento de los usuarios.(7)

XP

XP se clasifica dentro del grupo de las metodologías ágiles, la cual utiliza un enfoque orientado a objetos con el objetivo de lograr un producto en el menor tiempo posible, que sea fácil de usar y fiable. Esta metodología se basa en:

Pruebas Unitarias: son realizadas a los principales procesos con el objetivo de detectar futuros errores.

Re fabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. (8)

El uso de XP tiene entre sus principales ventajas realizar pruebas a los principales procesos con el objetivo de detectar futuros errores, posibilitando que los requisitos puedan ser modificados o cambiados, permitiéndole mayor adaptabilidad al producto, pues es especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Concibe que la propiedad del código es colectiva, cualquiera puede desarrollar, mejorar, simplificar, cualquier necesidad del

proyecto, siempre usando sistemas tipo CVS (Controlador de versiones). En esta metodología el cliente está presente en todo momento y colabora con el proyecto como un miembro más del mismo, existiendo de esta forma una comunicación efectiva en tiempo real entre el cliente y los desarrolladores, fortaleciendo el trabajo del proyecto en equipo.

OpenUP

OpenUP es una metodología ágil dirigida a la gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental apropiado para proyectos pequeños y de bajos recursos; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo. A continuación se definen: objetivos, disciplinas, fases y roles por los que la misma se rige. (9)

El objetivo de OpenUP es ayudar al equipo de desarrollo a través de todo el ciclo de vida de las iteraciones, de modo que este sea capaz de añadir valor de negocio para los clientes de una forma predecible: con la entrega de un software operativo y funcional al final de cada iteración. El ciclo de vida del proyecto provee a los clientes de una visión del proyecto, transparencia y les dota de los medios para que les permitan controlar la financiación, el riesgo, el ámbito y el valor de retorno esperado.

OpenUP se enfoca en las siguientes disciplinas:

- Requisitos: esta disciplina explica cómo identificar, analizar, especificar, validar y administrar los requisitos que debe cumplir el sistema.
- Arquitectura: esta disciplina explica cómo crear una arquitectura a partir de requisitos “estructuralmente” significativos. La arquitectura es construida en la disciplina de Desarrollo
- Desarrollo: esta disciplina explica como diseñar e implementar una solución técnica que cumple con la arquitectura y soporta los requerimientos de los grupos interesados.
- Prueba: esta disciplina explica como proveer retroalimentación al equipo de desarrollo acerca de la evolución del sistema a través del diseño, implementación, ejecución y evaluación de pruebas en cada uno de los componentes desarrollados.
- Gestión del Proyecto: esta disciplina tiene como objetivo ayudar a crear un ambiente de trabajo efectivo para maximizar la productividad del equipo.

- Administración de configuración y cambios: esta disciplina explica controlar el cambio en los artefactos, asegurando una evolución sincronizada del conjunto de productos de trabajo que componen el sistema software.

Fases:

- Concepción: primera de las 4 fases en el proyecto del ciclo de vida, acerca del entendimiento del propósito y objetivos y obteniendo suficiente información para confirmar que el proyecto debe hacer. El objetivo de esta fase es capturar las necesidades de los interesados (stakeholder) en los objetivos del ciclo de vida para el proyecto.
- Elaboración: es la segunda de las 4 fases del ciclo de vida del OpenUP donde se trata los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base para la elaboración de la arquitectura del sistema.
- Construcción: esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la arquitectura definida.
- Transición: es la última fase, cuyo propósito es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y rendimiento del último entregable de la fase de construcción.

Teniendo en cuenta la caracterización de estas tres metodologías de desarrollo de software, se seleccionó a OpenUP de la familia de metodologías de desarrollo de software del Proceso Unificado como la mejor opción para afrontar este proceso, pues es una metodología ágil, orientada al código y que se ajusta al ciclo de vida del proyecto debido a que es un proyecto pequeño que no necesita el desempeño de muchos roles.

1.4 Lenguaje de modelado

Un lenguaje de modelado constituye la forma común para establecer una coordinada comunicación entre los miembros de un equipo de desarrollo de software. Este lenguaje sirve de herramienta para desarrollar una representación simplificada de la realidad a través de abstracciones que se plasman en notaciones

gráficas y sirven de apoyo al análisis de problemas representándolo en forma más intuitiva para personas sin especialización en Informática.

UML

UML (por sus siglas en inglés Unified Modeling Language) es un lenguaje de modelado visual de propósito general orientado a objetos. Cubre las diferentes vistas de la arquitectura de un sistema mientras evoluciona a través del ciclo de vida del desarrollo de software.

Permite la representación conceptual y física de un sistema. UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de lo que se quiere representar. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. UML ofrece un estándar para describir un plano del sistema, incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. (10)

UML es un lenguaje de modelado visual que sirve para: visualizar, especificar, construir, documentar sistemas independientemente de la metodología de análisis y diseño pero siempre con una perspectiva orientada a objetos.

Ventajas de UML:

- Permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Permite especificar cuáles son las características de un sistema antes de su construcción.
- A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.
- Se basa en una notación gráfica concisa, fácil de aprender y utilizar.
- Se puede utilizar para modelar sistemas en diversos dominios: sistemas de informaciones empresariales, sistemas Web, sistemas críticos y de tiempo real, incluso en sistemas que no son software.

1.5 Herramientas CASE

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la calidad y la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero garantizando su fácil mantenimiento. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Visual Paradigm

Visual Paradigm ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Constituye una herramienta privada disponible en varias ediciones, cada una destinada a varias necesidades: Enterprise (Empresa), Professional (Profesional), Community (Comunidad), Standard, Modeler (Modelador) y Personal. Existe una alternativa libre y gratuita de este software, la versión Visual Paradigm UML 6.4 Community Edition. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque Orientado a Objetos. (11)

Dentro de sus características fundamentales se encuentran:

- Multiplataforma: es soportada en plataformas Java para sistemas operativos Windows, Linux y Mac OS X.
- Interoperabilidad: intercambia diagramas UML y modelos con otras herramientas. Soporta exportar e importar a XMI19, XML y archivos Excel. Importa archivos de proyectos de Rational Rose. posee integración con Microsoft Office Visio.
- Modelamiento de los Requisitos: permite captura de requisitos con diagrama de requisitos, modelamiento de casos de uso y análisis textual.
- Colaboración de Equipo: realiza el modelado simultáneamente con el Visual Paradigm Team Work Server y Subversion.
- Generación de Documentación: comparte y genera los diagramas y diseños en formatos como PDF, HTML y Microsoft Word.

- Ingeniería de Código: permite generación de código e ingeniería inversa en lenguajes como Java, C, PHP, XML, Python, C#, VB .NET, ActionScript, Delphi y Perl.
- Modelado de Procesos de Negocio: visualiza, comprende y mejora los procesos de negocio con la herramienta para estos procesos.
- Integración con Entornos de Desarrollo: apoyo el ciclo de vida completo de desarrollo del software: análisis, diseño e implementación, en IDE como Eclipse, Microsoft Visual Studio, NetBeans, Sun ONE, Oracle JDeveloper, JBuilder y otros.
- Modelamiento de Bases de Datos: generación de bases de datos, conversión de diagramas entidad-relación a tablas de base de datos, además de mapeos de objetos y relaciones entre tablas de base de datos.

1.6 Lenguajes de Programación y de Marcas

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; permitiendo al desarrollador comunicarse con los dispositivos de hardware y software existentes. (12)

PHP

PHP (por sus siglas en inglés Hypertext Pre Processor) es un lenguaje script del lado del servidor que es embebido dentro del código HTML utilizado para la generación de páginas Web dinámicas. Es un lenguaje sencillo de sintaxis cómoda, es rápido y dispone de una gran cantidad de librerías que facilitan el desarrollo de aplicaciones, es de código abierto, lo cual significa que el usuario no depende de una licencia específica para arreglar cosas que no funcionan, además no está forzado a pagar actualizaciones anuales para tener una versión del producto. (13)

La versión utilizada para el desarrollo de la aplicación es PHP 5.3 que incluye varias características tales como:

- Soporte para espacios de nombres (namespaces).
- Mejor soporte de Windows, así como la portabilidad a otras plataformas soportadas.
- Mejor soporte para la Programación Orientada a Objetos, en versiones anteriores era extremadamente rudimentario.
- Mejoras de rendimiento.

- Mejor soporte a XML.
- Manejo de excepciones. (13)

XML

XML (Extensible Markup Language, Lenguaje de Marcado Extensible), una especificación/lenguaje de programación desarrollada por el W3C[5]. Permite que los diseñadores creen sus propias etiquetas, permitiendo la definición, transmisión, validación e interpretación de datos entre aplicaciones y entre organizaciones. (14)

XML consiste en un conjunto de reglas para representar información en una forma fácilmente procesable por un ordenador. Su diseño persigue enfatizar la simplicidad y usabilidad a través de Internet. Su formato es basado en caracteres de texto con soporte para todos los idiomas. Originalmente diseñado para afrontar el reto que representa la publicación de información electrónica a gran escala, XML ha jugado un gran papel en el intercambio de información en la Web y otras esferas.

Posee las siguientes características: (14)

- **Es un estándar para escribir datos estructurados en un fichero de texto.** Provee un conjunto de reglas, normas y convenciones para diseñar formatos de texto para datos estructurados que van desde las hojas de cálculo, o las libretas de direcciones de Internet, hasta parámetros de configuración, transacciones financieras o dibujos técnicos. Los programas que los generan, utilizan normalmente formatos binarios o de texto. Permite resolver problemas comunes, como la falta de extensibilidad, carencias de soporte debido a características de internacionalización, o problemas asociados a plataformas específicas.

- **XML parece HTML pero no lo es.** Tanto XML como HTML usan marcas y atributos, más su diferencia radica en que, en HTML cada marca y atributo establece un significado a la vez incluyendo el aspecto que debe tener al verse en un navegador, en XML sólo se usan las marcas para delimitar fragmentos de datos, dejando la interpretación de estos a la aplicación que los lee.

[5] W3C son las siglas de World Wide Web Consortium, un consorcio fundado en 1994 para dirigir a la Web hacia su pleno potencial mediante el desarrollo de protocolos comunes que promuevan su evolución y aseguren su interoperabilidad.

- **XML está en formato texto, pero no para ser leído.** El formato texto puede ser usado en cualquier plataforma, esto le da innumerables ventajas de portabilidad, depuración, independencia de plataforma, e incluso de edición, pero su sintaxis es más estricta que la de HTML: una marca olvidada o un valor de atributo sin comillas convierten el documento en inutilizable.

- **XML no requiere licencia.** Es un estándar abierto independiente de la plataforma, y tiene un amplio soporte extendido a un sinnúmero de herramientas y desarrolladores.

JavaScript

Es 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. Es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos, se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. Los programas en JavaScript van incluidos en los documentos XHTML o en un fichero js y se encargan de realizar acciones en el cliente, como puede ser: pedir datos, confirmaciones, mostrar mensajes, crear animaciones y comprobar campos. (15)

Ventajas:

- Los programas escritos en este lenguaje no requieren de mucha memoria ni tiempo adicional de transmisión, por ser pequeños y compactos.
- No requiere un tiempo de compilación, ya que los script se pueden desarrollar en un periodo de tiempo relativamente corto.
- Es independiente de la plataforma hardware o sistema operativo, funciona correctamente siempre y cuando exista un navegador que lo soporte.
- Asegura la permanencia de una operación realizada y aunque falle el sistema esta no podrá deshacerse.

1.7 Marco de trabajo

Dentro del ambiente de desarrollo de software, un marco de trabajo es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, sirviendo de base para que otro proyecto de software pueda ser fácilmente organizado y desarrollado. Puede incluir soporte de programas, librerías y otras herramientas para ayudar a desarrollar y unir los diferentes

componentes de un proyecto. Los principales objetivos de los marcos de trabajo son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. El cumplimiento de estos objetivos, facilita el desarrollo del software, permitiendo a diseñadores y programadores concentrarse en los requerimientos del proyecto reduciendo los posibles problemas con las tecnologías utilizadas, así como facilitando ciertas funcionalidades básicas y comunes.

Symfony 1.1

Symfony es un marco de trabajo bajo licencia MIT que sigue el patrón arquitectónico Modelo-Vista-Controlador para desarrollar aplicaciones web comerciales, gratuitas y/o de software libre escritas en PHP5. Tiene como objetivo acelerar la creación y mantenimiento de aplicaciones web y sustituir repetitivas tareas de codificación; proporcionando para esto varias herramientas que siguen la mayoría de las mejores prácticas y patrones de diseño para la web. Este marco de trabajo es fácil de instalar y extender, lo que permite su integración con librerías desarrolladas por terceros entre ellas Zend Framework. Presenta también herramientas de configuración que lo hacen lo suficientemente flexible como para adaptarse a las necesidades de la aplicación que se desea desarrollar. Otras de las características de Symfony es que brinda soporte a la internacionalización y es independiente del sistema gestor de base de datos, algo que se garantiza a través de un ORM[6] (Propel o Doctrine). Las aplicaciones que emplean Symfony, pueden controlar hasta el último acceso a la información, debido a que el framework brinda herramientas para ello, e incluye por defecto protección contra ataques XSS[7].

Ext JS 2.2

Ext JS es una librería de Java Script para el desarrollo de aplicaciones web enriquecidas, haciendo un uso intensivo de las tecnologías AJAX, XHTML/ DHTML y DOM .Originalmente fue creado como una extensión de Yahoo User Interface (YUI) otro framework similar, Ext incluye interoperabilidad con jQuery, Prototype y Scrpt.aculo.US.

[6] **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relational mapping ORM) es una técnica de programación para convertir datos entre el lenguaje de programación orientado a objetos utilizado y el sistema de base de datos relacional utilizado en el desarrollo de una aplicación.

[7] **XSS Secuencias de comandos en sitios cruzados**,(del inglés ***Cross-site scripting***)es un tipo de inseguridad informática o agujero de seguridad típico de las aplicaciones Web, que permite a una tercera parte inyectar en páginas web vistas por el usuario código JavaScript o en otro lenguaje script similar.

Sus principales características son:

- Alto rendimiento en ejecución debido a la optimización de código Java Script.
- Controles de usuario personalizables.
- Modelo orientado a componentes, bien diseñado y extensible.
- Posee una API intuitiva y fácil de utilizar.
- Es distribuido bajo licencias OpenSource y comerciales.

La versión 2.2 entre sus mejoras incluye:

- Soporte para peticiones directas, CRUD y REST.
- Nuevos ejemplos y componentes (incluye un componente para gráficas).
- Administración de memoria mejorada para Internet Explorer 6.
- API documentada.
- Compatibilidad con versiones anteriores.

1.8 Herramientas de Desarrollo

Un entorno de desarrollo integrado o IDE (Integrated Development Environment), es un programa informático compuesto por un conjunto de herramientas de programación. Estos sistemas informáticos han sido empaquetados como programas de aplicaciones y generalmente están compuestos por un conjunto de herramientas de programación: un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Pueden manejar uno o varios lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic.

NetBeans

El IDE NetBeans es un entorno galardonado de desarrollo integrado disponible para Windows, Mac, Linux y Solaris. El proyecto NetBeans consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear aplicaciones web, de escritorio y aplicaciones móviles utilizando la plataforma Java, así como PHP, JavaScript, Ajax, Groovy y Grails, y C / C + +. El proyecto de NetBeans está apoyado por una comunidad de desarrolladores y ofrece una amplia documentación y recursos de capacitación, así como una gran cantidad de plugins de terceros.

NetBeans funciona en sistemas operativos compatibles con la máquina virtual de Java (Windows XP, Windows Vista, Windows 7, Ubuntu, Solaris, Mac OS X 10.5 o superior). En su versión 6.5 se ha incorporado el lenguaje de trabajo PHP y en su versión actual se ha incluido a Synfony como framework por defecto.

NetBeans IDE 7.2 es la versión que se utilizará, este ofrece un rendimiento significativamente mejorado, aparece con nuevas capacidades de análisis de código estático en el Editor de Java y más eficaz en la exploración del proyecto. El lanzamiento también incluye características notables, como la integración con el Generador de Escena para la creación de formas visuales JavaFX, soporta marcos de PHP múltiples y muchas otras mejoras en Java EE, Maven, C / C + + y la plataforma NetBeans. (16)

NetBeans IDE 7.2 incluye los siguientes cambios:

- Aumento de rendimiento significativo en los sistemas de archivos remotos.
- Mayor velocidad de escaneo.
- Mejoras del Editor Java.
- Mejoras del depurador.
- Soporta Symfony2, Doctrine2 y marcos ApiGen.
- Compatible con PHP 5.3.
- Depuración de código y mejoras de asistencia en proyectos de C + +.

NetBeans IDE 7.2 soporta las siguientes tecnologías y ha sido probado con los siguientes servidores de aplicaciones.

Tecnologías compatibles:

- Java EE 6, Java EE 5 y J2EE 1.4.
- JavaFX 2.2.
- Java ME SDK 3.2.
- Java Card 3 SDK.
- Hibernate 3.2.5.
- PHP 5.3.
- Apache Ant 1.8.3.
- Apache Maven 3.0.4 y anteriores.
- C / C + + / Fortran.

- Mercurial: 2.0.x y anteriores.

Conocido por ejecutar servidores de aplicaciones:

- GlassFish Server Open Edition 3.0.1.
- Tomcat 7.0.27.
- JBoss 6.1.

Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma, es una herramienta para desarrolladores Java que permite crear aplicaciones JEE y web, incluye un entorno integrado de desarrollo para Java. Posee una estructura modular, extensible mediante plugins, que le permite trabajar con cualquier tipo de recurso: gráficas, vídeo, modelos 3D, contenido web. Otros lenguajes que también pueden utilizarse en Eclipse son: C/C++, PHP, Ruby, TCL o JavaScript. Como IDE para Java cuenta algunas funciones interesantes, entre ellas: desarrollo de aplicaciones en grupo, unidad integrada de depuración y pruebas, compilación y construcción incremental. Eclipse es una gran estructura formada por un núcleo y muchos plug-ins que van conformando la funcionalidad final. La forma en que los plug-ins interactúan es mediante interfaces o puntos de extensión; así las nuevas portaciones se integran sin dificultad ni conflictos. Todo este conjunto de características lo convierten en una plataforma de herramienta universal, abierta y extensible para el desarrollo del software.

A partir del estudio realizado de estas dos herramientas de desarrollo, se decide seleccionar el NetBeans en su versión 7.2 para el desarrollo de la solución, debido a sus nuevas capacidades de análisis de código estático, sus características de ser libre y la riqueza de sus prestaciones.

1.9 Servidor de aplicaciones

Un servidor de aplicaciones proporciona servicios que soportan la ejecución y disponibilidad de las aplicaciones desplegadas. Es el núcleo de un gran sistema distribuido y gestiona gran parte o la totalidad de las funciones de lógica de negocio y de acceso a los datos de una aplicación. Como principal beneficio de esta tecnología se encuentra: la centralización y la disminución de la complejidad en el desarrollo de aplicaciones.

Servidor Web Apache 2.2

El servidor Apache es desarrollado por la Apache Software Foundation, es un servidor Web que se carga como un servicio más del sistema operativo. Cuando está activo, convierte la computadora en un servidor capaz de enviar contenido a cualquier navegador. Presenta entre otras características:

- Mensajes de error altamente configurables, bases de datos de autenticación y negociado de contenido.
- Permite la creación de sitios web dinámicos mediante el uso de Server Side Includes (SSI) (por sus siglas en inglés), de lenguajes de scripting como PHP, JavaScript, Python entre otros.
- Se ejecuta en varios sistemas operativos.
- Posee una arquitectura modular que le permite ser adaptado a diferentes entornos y necesidades, con los diferentes módulos de apoyo que proporciona.
- Es eficiente y flexible; permitiendo personalizar la respuesta ante los posibles errores que se pueden generar en el servidor.

Entre sus principales ventajas se encuentran: posee gran cantidad de extensiones para diversas tecnologías, además de una amplia documentación, es libre, modular y multiplataforma. Se señala como desventaja que no posee interfaz gráfica que facilite su configuración.

1.10 Conclusiones del capítulo

En el presente capítulo se plantearon los principales conceptos relacionados con el objeto de estudio donde se realizó un estudio detallado sobre las etapas de desarrollo de un intérprete así como sus fases para tener claridad de cómo comenzar el desarrollo de dicho intérprete. Se seleccionó como metodología de desarrollo de software OpenUP. Se decidió utilizar como herramienta de modelado Visual Paradigm 6.4 y como lenguaje de modelado UML. Para la implementación de la solución se utilizarán los lenguajes de programación: PHP 5.3 y JavaScript soportados por los marcos de desarrollo ExtJS 2.2 y Symfony 1.1, XML como lenguaje de marcas extensible, como herramienta de desarrollo NetBeans 7.2, y como servidor web para correr código PHP Apache 2.2. Estas herramientas apoyan al desarrollo general de la solución propuesta, brindando agilidad y variedad de funciones a los desarrolladores, mencionando además que son herramientas libres.

CAPÍTULO II: ANÁLISIS Y DISEÑO DEL SISTEMA

Introducción

En el presente capítulo se describe el proceso llevado a cabo durante la fase de análisis y diseño. Se definen los requisitos funcionales y no funcionales del software lo cual es un paso de gran importancia, para su éxito. Se muestra la estructura del sistema a desarrollar a través de una representación visual de las clases conceptuales en el modelo de dominio. Se especifica la estructura física de la solución que se propone mediante el diagrama de casos de uso del sistema y el modelo de despliegue. Se muestran los diagramas de clases del diseño, por medio de los cuales se representa la estructura estática del sistema.

2.1 Modelo de Dominio

El Modelo de Dominio es una representación visual estática donde se modelan los términos más relevantes del entorno donde se desarrolla el software y los elementos relacionados con este. El objetivo del modelado del dominio es contribuir a la comprensión de los conceptos clave de un negocio o un dominio del problema. (17)

A continuación se presenta en la Figura 2 el Modelo de Dominio del Intérprete SQL y se describen sus clases.

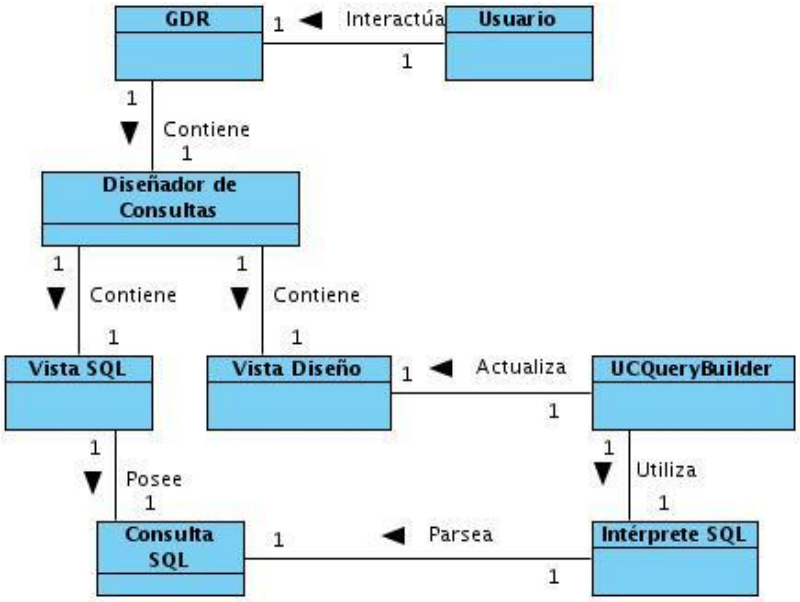

Figura 2 Modelo de Dominio.

Descripción de las clases del dominio:

Usuario: en este caso se comporta como el diseñador de consulta que interactúa con el sistema Generador Dinámico de Reportes específicamente con el módulo Diseñador de consulta.

GDR: entidad que representa al sistema Generador Dinámico de Reportes.

Diseñador de consulta: representa uno de los módulos que contiene el sistema GDR, el mismo funciona como un editor gráfico de consultas SQL permitiendo crear, editar y ejecutar consultas del tipo antes mencionado.

Vista de diseño: vista perteneciente al módulo Diseñador de consulta, en esta se realiza el diseño gráfico de las consultas.

Vista SQL: vista perteneciente al módulo Diseñador de consulta, donde se muestra el código fuente perteneciente a la consulta en diseño, esta permite realizar cambios a dicho código.

Consulta SQL: representa al código SQL contenido en la Vista SQL que será verificado por el intérprete.

UCQueryBuilder: es la clase controladora principal del módulo, contiene las principales operaciones que se realizan del lado del cliente.

Intérprete SQL: representa al parse encargado de verificar la estructura y sintaxis del código SQL contenido en la Vista SQL.

2.2 Requisitos del sistema

El propósito fundamental de la captura de los requisitos es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción clara y precisa de los requisitos del sistema que determine un acuerdo entre el cliente (incluyendo a los usuarios) y los desarrolladores sobre qué debe y qué no debe hacer el sistema. En otras palabras los requisitos del sistema son cualidades o capacidades que debe poseer un sistema para satisfacer a un cliente, un contrato u otra documentación formalmente impuesta.

2.2.1 Requisitos funcionales

Los requisitos funcionales pueden ser una descripción de lo que un sistema ha de cumplir para su correcto funcionamiento. Este tipo de requisito específica algo que el sistema entregado debe ser capaz de realizar para cumplir con las especificaciones de software. Para cumplir los objetivos de este sistema el mismo debe ser capaz de:

RF1. Editar el código de una consulta SQL en diseño.
Descripción: permitirá modificar desde la Vista SQL la consulta que se está diseñando.
Entrada: cambios en la estructura de la consulta SQL.
Salida: consulta SQL modificada en la Vista correspondiente.

RF2. Actualizar el diseño de una consulta SQL.
Descripción: el sistema verificará que la sentencia de la consulta este correctamente y validará que los datos de la misma sean correctos, posteriormente permitirá actualizar en la Vista de diseño la consulta modificada.
Entrada: consulta SQL en formato XML.
Salida: diseño actualizado en la Vista correspondiente.

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener, debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. El Diseñador de Consultas es un módulo del producto Generador Dinámico de Reportes, por lo que debe poseer los requerimientos no funcionales definidos en la arquitectura del mismo. Algunos de ellos se describen a continuación:

RNF 1. Usabilidad

La herramienta se desarrolla para la Web, pero con características muy similares a las aplicaciones de escritorio, en cuanto al diseño de las interfaces visuales y los tiempos de respuesta, por lo que el usuario debe poder diseñar una consulta SQL de manera ágil y sencilla, sin ser un experto en el proceso de diseño de un reporte.

RNF 2. Interfaz

Las interfaces de usuario serán diseñadas a modo de aplicaciones RIA (Rich Internet Application) lo que permite a los usuarios contar con aplicaciones web con una experiencia de usuario similar a la de las aplicaciones de escritorio. Para lograr este fin se usará la librería JavaScript ExtJS, la cual conjuga una serie de componentes visuales que proveen funcionalidades que ayudan al diseño de este tipo de aplicaciones Web con apariencia de escritorio.

RNF 3. Restricciones de Diseño e Implementación

El sistema deberá ser implementado en el lenguaje de programación PHP versión 5.3, como framework de desarrollo la librería de JavaScript ExtJS versión 2.2 y Symfony 1.1. Como herramienta de desarrollo se usará NetBeans 7.2.

RNF 4. Software

El servidor donde se instalará la aplicación debe cumplir con los siguientes requisitos:

Sistema Operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 8.04 o superior, Debian 4.0 GNU/Linux o superior.

Paquetes: apache2, php5, libapache2-modphp5, php5-cli, php5-mysql, php5-pgsql, php5-sqlite, php5-xsl, php5-gd.

Usuario con privilegios de administración.

El servidor donde se instalará la base de datos debe cumplir con los siguientes requisitos:

Sistema Operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 8.04 o superior, Debian 4 GNU/Linux o superior.

PostgreSQL versión 8.3, 8.4.

PGAdmin III o algún administrador para PostgreSQL.

Usuario con privilegios para instalar la base de datos.

PostgreSQL debe estar correctamente configurado para aceptar conexiones vía TCP/IP.

La PC utilizada para acceder a la aplicación debe cumplir con los siguientes requisitos:

Sistema Operativo:GNU/Linux preferentemente Ubuntu GNU/Linux 8.04 o superior, Debian 4 GNU/Linux o superior, Microsoft Window XP o superior.

Navegador Web Mozilla Firefox 2.0 o superior.

RNF 5. Hardware

El servidor donde se instalará la aplicación debe cumplir con los siguientes requisitos:

Procesador Intel Pentium 4 1.7 GHz o AMD similar.

512 MB de RAM.

40 GB de espacio en disco duro.

El servidor donde se instalará la base de datos debe cumplir con los siguientes requisitos:

Procesador Intel Pentium 4 1.7 GHz o AMD similar.

512 MB de RAM.

40 GB de espacio en disco duro.

La PC utilizada para acceder a la aplicación debe cumplir con los siguientes requisitos:

Procesador Intel Pentium 4 1.7 GHz, o AMD similar.

256 MB RAM.

2.3 Modelo de casos de uso del sistema

Un modelo de casos de uso es un modelo del sistema que contiene actores, casos de uso y sus relaciones. El modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema. Los casos de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores, estos especifican una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia. Los casos de uso son el componente clave del modelado. Su propósito es ilustrar como un sistema permite a un actor cumplir una meta, ilustrando todos los posibles caminos apropiados que ellos pueden tomar para cumplirla, así como las situaciones que podrían hacerlo fallar.

2.3.1 Diagrama de casos de uso del sistema (DCUS)

El diagrama de casos de uso documenta el comportamiento de un sistema desde el punto de vista del usuario. Por lo tanto los casos de uso determinan los requisitos funcionales del sistema, es decir, representan las funciones que un sistema puede ejecutar. Para el modelado del diagrama de casos de uso del sistema, se tuvo en cuenta el uso de patrones de casos de uso. Estos patrones son un conjunto de comportamientos que deben existir en el sistema, ayudan a describir qué es lo que el sistema debe hacer y cómo este interactúa con los usuarios. Generalmente son utilizados como plantillas que describen cómo debería ser estructurados y organizados los casos de uso, capturando así mejores prácticas para modelar casos de uso. Entre estos patrones, el que se evidencia en el diagrama de casos de uso del sistema es: Concreta inclusión, este consiste en dos casos de uso y una relación de inclusión entre ellos, es decir, se incluye una relación del caso de uso base (Modificar consulta SQL) al caso de uso de inclusión (Actualizar diseño). A continuación la Figura 3 muestra el diagrama de casos de uso del sistema:

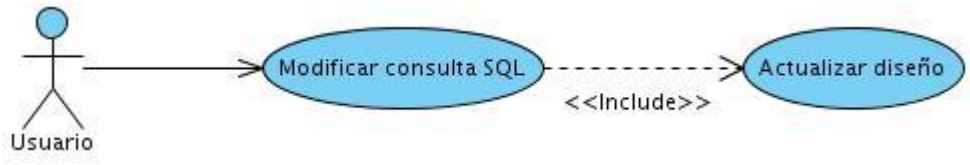

Figura 3 Diagrama de casos de uso del sistema.

Descripción de los casos de uso:

Modificar consulta SQL: permite modificar la consulta SQL que se está diseñando.

Actualizar diseño: permite actualizar el diseño de la consulta SQL modificada.

2.3.2 Descripción del Caso de Uso arquitectónicamente significativo del sistema.

Caso de Uso:	Actualizar diseño	
Actores:	Usuario	
Resumen:	El caso de uso se inicia cuando el actor modifica la consulta en diseño desde la Vista SQL, posteriormente el sistema verifica que la consulta esté correctamente estructurada y muestra una interfaz con el diseño de la consulta modificada. El caso de uso termina una vez que se haya actualizado la Vista de diseño.	
Precondiciones:	El sistema debe tener al menos una consulta cargada para ser modificada.	
Referencias	RF1, RF2	
Prioridad	Crítico	
Flujo Normal de Eventos		
Sección “”		
Acción del Actor		**Respuesta del Sistema**
1. Selecciona la pestaña Vista SQL.		2. Muestra en una interfaz la consulta SQL en diseño.
3. Modifica la consulta y selecciona la pestaña Vista de diseño.		4. Verifica que la consulta esté correctamente estructurada y muestra

	una interfaz con el diseño de la consulta modificada. En caso de que la consulta esté incorrecta ver el *Flujo Alterno 1* Terminando así el caso de uso.
Prototipo de Interfaz	
Flujo Alterno al paso 4 "Error"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un mensaje de error.
Prototipo de Interfaz	
Poscondiciones:	Se actualiza el diseño de la consulta modificada en la Vista de diseño.

2.4 Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en cómo los requisitos funcionales y no funcionales junto con otras restricciones relacionadas con el entorno de implementación tienen impacto en el sistema a considerar. El modelo de diseño sirve de abstracción a la implementación y es utilizado como una entrada fundamental a las actividades de implementación.

2.4.1 Diagrama de clases del diseño

En este diagrama se muestran las clases participantes en la ejecución de cada caso de uso, subsistemas y sus relaciones. También puede darse el caso de que se representen algunas operaciones, atributos y asociaciones sobre una clase específica que son relevantes. A continuación la Figura 4 muestra el diagrama de clases del diseño correspondiente al caso de uso Actualizar diseño:

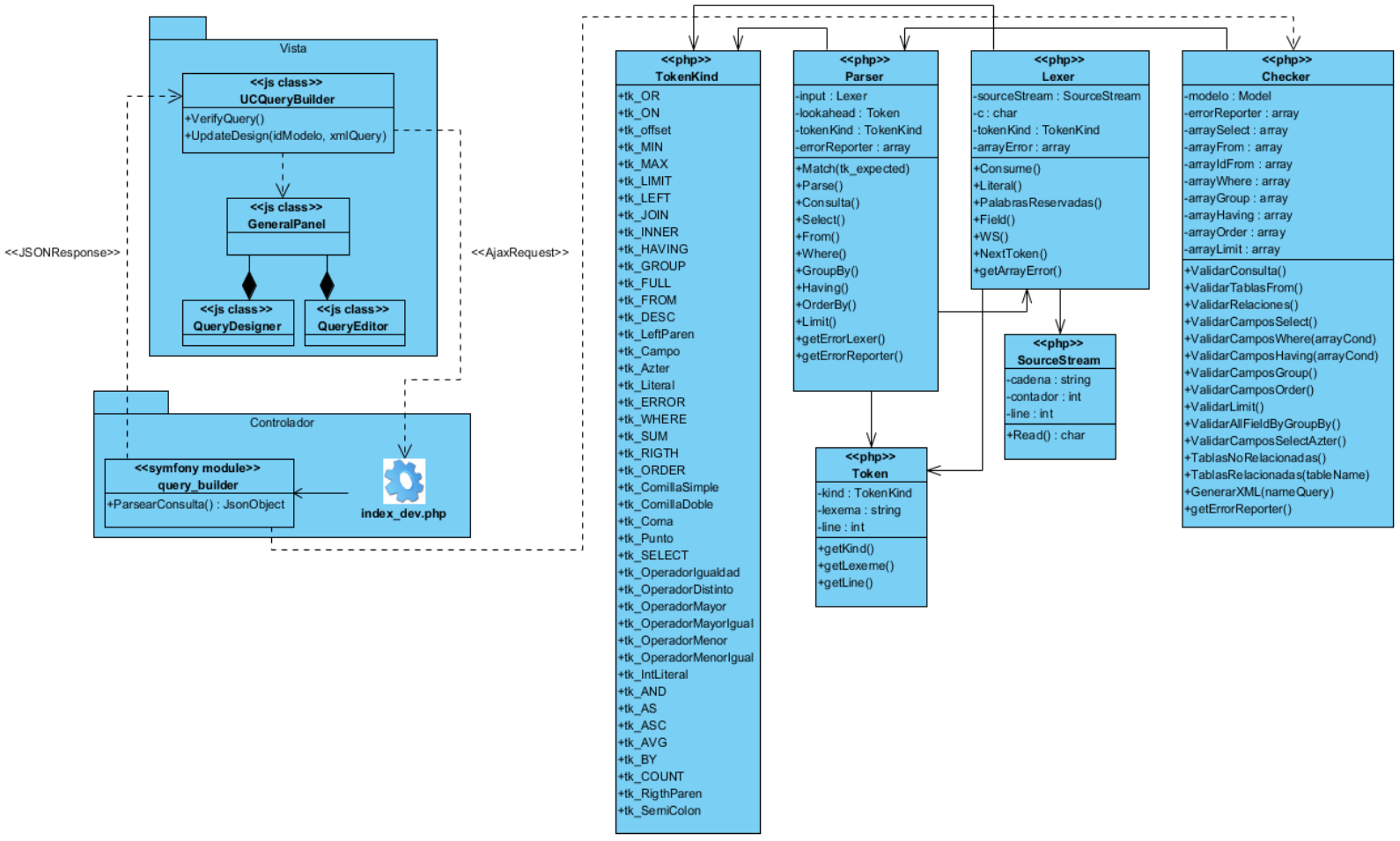

Figura 4 Diagrama de clases del diseño del CU Actualizar diseño

Descripción de las clases

En el diagrama de clases del diseño anterior se representaron las principales clases y métodos que se utilizaron para la realización del caso de uso Actualizar diseño, a continuación serán descritas dichas clases:

UCQueryBuilder: es la clase controladora principal del módulo, contiene las principales operaciones que se realizan del lado del cliente, entre estas operaciones se encuentra VerifyQuery(), implementada como la acción que se encarga de obtener la consulta SQL modificada y de verificar si está correctamente

estructurada, otra función perteneciente a esta clase es UpdateDesign() encargada de actualizar el diseño una vez que se haya comprobado que la sintaxis de la consulta SQL este correctamente estructurada.

Lexer: desglosa la consulta SQL en diferentes tokens identificando así palabras reservadas propias del lenguaje SQL, sería tomar cada uno de los caracteres de la consulta y agruparlos en entidades sintácticas simples o elementales denominadas tokens, para crear cada tipo de tokens esta clase hace uso de las clases Token contenedora de la información de este objeto indicando en que línea se encuentra el token, el tipo de token, y que lexema posee, también se hace uso de la clase SourceStream indicando el caracter presente de la consulta analizada y la clase TokenKind que indica el tipo de token.

Parser: comprueba si la estructura de la consulta modificada cumple con la estructura definida por la gramática, a este proceso se le denomina análisis sintáctico en el cual se examina una secuencia de tokens para determinar si el orden de esa secuencia es correcto de acuerdo a ciertas convenciones estructurales o reglas de la definición sintáctica del lenguaje. La entrada del analizador sintáctico o parser es la secuencia de tokens generada por el analizador léxico (Lexer) y la salida es la indicación del cumplimiento de las reglas gramaticales que definen al lenguaje.

Cheker: permite comprobar semánticamente la consulta SQL, esta es la encargada de detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico (Parser) comprobando las reglas semánticas del lenguaje.

2.4.2 Diagrama de secuencia

Un diagrama de secuencia muestra una interacción que está organizada como una secuencia temporal. En particular, muestra los objetos que participan en la interacción mediante sus líneas de vida y los mensajes que intercambian, organizados en forma de una secuencia temporal.
A continuación la Figura 5 muestra el diagrama de secuencia del caso de uso Actualizar diseño:

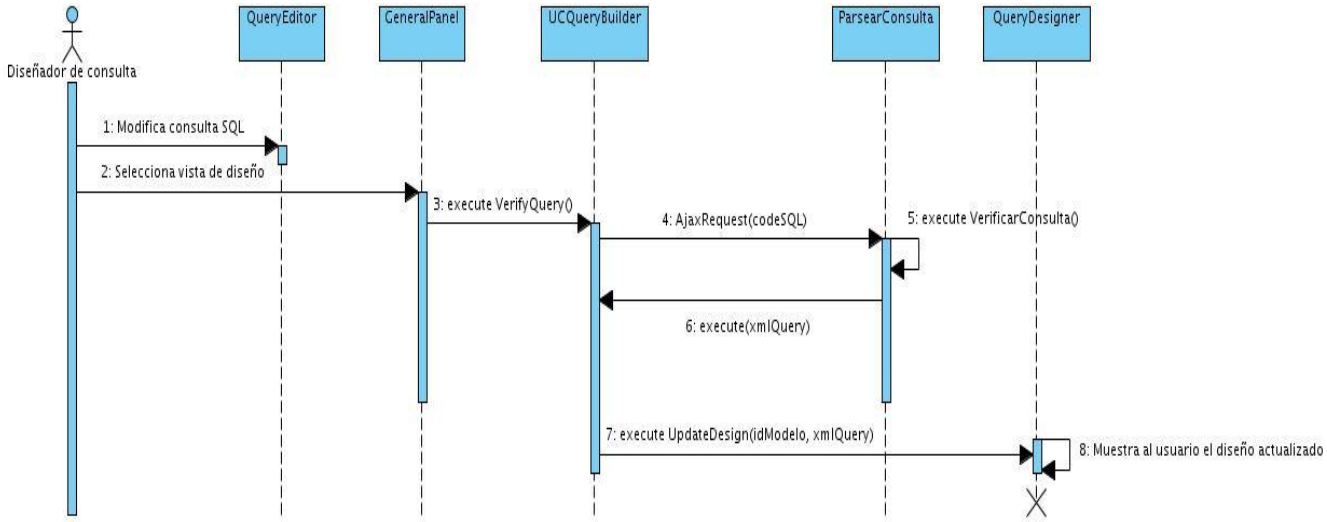

Figura 5 Diagrama de secuencia del CU Actualizar diseño

Como se representa en el diagrama de la Figura 5, el actor modifica la consulta contenida en la Vista SQL y selecciona la pestaña Vista de diseño, luego la página cliente se encarga de mandar a ejecutar el método VerifyQuery() en la clase UCQueryBuilder, esta llamada permite enviar la petición de comprobar si la consulta está correctamente estructurada a la clase ParsearConsulta.php ejecutando el método ValidarConsulta() de la clase Cheker.php, posteriormente la clase ParsearConsulta.php envía una respuesta a la acción, la acción captura la respuesta del controlador y en caso de que la consulta este correctamente estructurada se ejecuta el método UpdateDesign() en la clase UCQueryBuilder, esta llamada permite actualizar la Vista de diseño y finalmente el sistema le muestra al usuario el diseño actualizado.

2.5 Patrones utilizados en la solución

Un patrón se define como una solución probada con éxito que aparece una y otra vez ante determinado tipo de problema en un contexto dado. Según el criterio de Microsoft “los patrones se refieren más bien a prácticas de re-utilización y se encuentran más ligados al uso y al plano físico”.

2.5.1 Patrón Arquitectónico

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen

reglas y guías para organizar las relaciones entre ellos. Los patrones de arquitectura representan el nivel más alto en el sistema de patrones. Ayudan a especificar la estructura fundamental de una aplicación. Cada actividad de desarrollo es gobernada por esta estructura; por ejemplo, el diseño detallado de los subsistemas, la comunicación y colaboración entre diferentes partes del sistema. Cada patrón de arquitectura ayuda a conseguir una propiedad específica en el sistema global. (18)

En la arquitectura del presente trabajo está presente el siguiente patrón arquitectónico.

Modelo - Vista - Controlador

Es un patrón que separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes: Modelo, Vista y Controlador. (19)

Modelo: El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista: Maneja la visualización de la información.

Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado. (Ver figura 6)

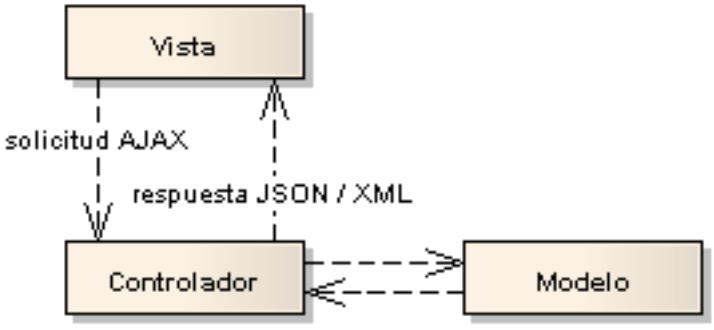

Figura 6 Patrón Arquitectónico Modelo Vista Controlador (19)

2.5.2 Patrones de Diseño

Un patrón de diseño es una solución estándar para un problema común de programación. Son una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios y un proyecto o estructura de

implementación que logra una finalidad determinada. Entre sus principales características se encuentra que favorecen la reutilización de código que ayudan a construir software basado en la reutilización (a construir clases reutilizables), los propios patrones se reutilizan cada vez que se vuelven a aplicar. (20)

Patrones GRASP

Los patrones GRASP son parejas de problema-solución con un nombre, que codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades. Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

Patrón Experto: Este patrón se encarga de asignar una responsabilidad al experto en información, la clase que cuenta con la información necesaria para cumplir la responsabilidad. En el presente trabajo se hace uso del mismo en la clase Parser.php que es la encargada de realizar el análisis sintáctico. Esta clase es la indicada para llevar a cabo esta responsabilidad debido a la estructura y funcionalidades que posee. (Ver figura 7)

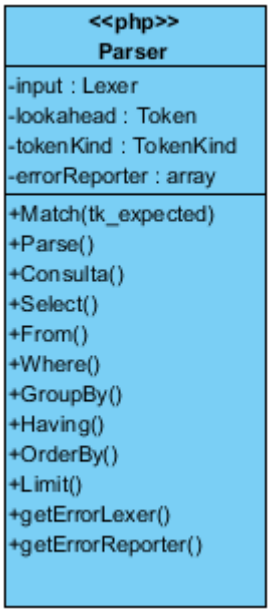

Figura 7 Ejemplo del patrón Experto

Patrón Alta Cohesión: Mantiene la complejidad dentro de límites manejables, es decir asigna una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Con el uso de este patrón se garantiza que la información que almacena una clase sea coherente y esté relacionada con la clase. (20)
Este patrón se evidencia en la clase Lexer.php, encargada de tomar cada uno de los caracteres de la consulta y agruparlos en entidades sintácticas simples o elementales denominadas tokens, debido a estas características esta clase carga con la responsabilidad de realizar el análisis léxico. (Ver figura 8)

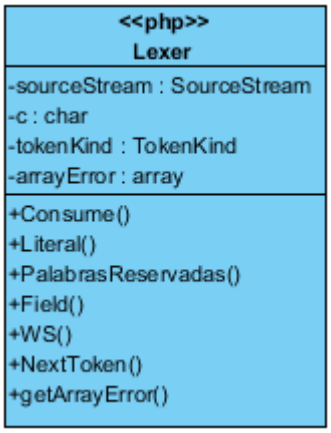

Figura 8 Ejemplo del patrón Alta Cohesión

2.5.3 Patrones de diseño GoF

Los patrones de diseño GoF, son clasificados según su propósito en creacionales, estructurales y de comportamiento, mientras que respecto a su ámbito se clasifican en clases y objetos. En el presente trabajo se utilizaron los patrones estructurales y a continuación se explica en qué consiste estos patrones.

Patrones Estructurales: Separan la interfaz de la implementación, se ocupan de cómo las clases y objetos se agrupan para formar estructuras más grandes. Dentro de estos se encuentran numerosos tipos como por ejemplo: Adaptador (Adapter), Decorador (Decorator), Fachada (Facade), Puente (Bridge), Objeto Compuesto (Composite), Peso Ligero (Flyweight), Apoderado (Proxy). En la solución se utiliza el patrón Fachada (Facade), el mismo simplifica el acceso a un conjunto de clases proporcionando una única clase que todos utilizan para comunicarse con dicho conjunto. Este patrón se evidencia en la clase

GeneralPanel la cual se comporta como mediadora en la comunicación con otras clases que facilitan el desarrollo de la solución.

2.6 Diagrama de Despliegue

Un diagrama de despliegue muestra las relaciones físicas entre los componentes de hardware y software en el sistema final, es decir la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software. Mediante el diagrama de despliegue se captura la configuración de los elementos de procesamiento, sus conexiones y se visualiza la distribución de los componentes de software en los nodos físicos.

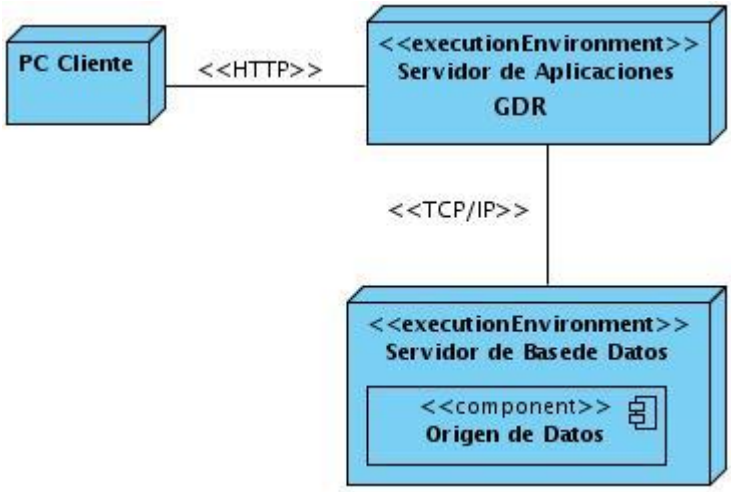

Figura 9 Diagrama de Despliegue

El diagrama de despliegue realizado representa tres nodos principales. El nodo PC_Cliente que requiere de un navegador que soporte Java Script, el nodo Servidor de Bases de Datos desde donde se cargará el Origen de Datos y el nodo Servidor de Aplicaciones, en el cual debe estar instalado el servidor web.

El nodo PC Cliente estará conectado al nodo Servidor de Aplicación por el protocolo de transferencia de hipertexto HTTP (por sus siglas en inglés Hypertext Transfer Protocol) como protocolo de comunicación. La conexión entre el Servidor de Aplicaciones y el Servidor de Base de Datos se realizará mediante el protocolo de comunicación TCP/IP.

2.7 Conclusiones del capítulo

En el presente capítulo fueron descritos el análisis y el diseño del sistema comenzando por el modelo de dominio, los requisitos funcionales y no funcionales. Además se identificaron los actores, casos de uso y la relación existente entre ellos reflejada en el diagrama de casos de uso del sistema, apoyado de la descripción detallada de los mismos. Partiendo de la realización de los casos de uso, se realizó el diagrama de secuencia, reflejando un proceso lógico de acciones para resolver las peticiones del cliente. A través del diagrama de despliegue se describió como se realizará el despliegue del módulo Diseñador de Consulta a lo largo de la infraestructura del sistema, para así dar comienzo a la implementación de la aplicación.

Capítulo III: Implementación y Prueba

Introducción

En el presente capítulo se define la gramática que guiará el proceso de interpretación y se realiza la especificación del lenguaje a interpretar. Se muestra el modelo de implementación que pone en práctica el diseño de la solución que se va a realizar, así como el diagrama de componentes del intérprete SQL. Se diseñan las pruebas que se le aplicarán al sistema apoyándose en los casos de pruebas para comprobar el correcto funcionamiento de las principales funcionalidades de la aplicación.

Gramática y Especificación del lenguaje a interpretar

Antes de implementar un intérprete para un lenguaje determinado, es necesario definir su gramática y especificación. Para ello hay que identificar primeramente la colección de estructuras léxicas que componen el lenguaje que será interpretado; para los identificadores, literal de cadena y literal entero el patrón se representó usando autómatas finitos. A continuación se muestra una tabla con los diferentes tipos de token en que se agruparán la secuencia de símbolos de la entrada o código fuente, los posibles lexemas que pueden presentar las entidades sintácticas simples o elementales y las categorías de cada token.

Tipo de Token	Lexema	Categoría
tk_SELECT	SELECT	Palabra reservada
tk_AS	AS	Palabra reservada
tk_FROM	FROM	Palabra reservada
tk_INNER	INNER	Palabra reservada
tk_RIGTH	RIGTH	Palabra reservada
tk_LEFT	LEFT	Palabra reservada
tk_FULL	FULL	Palabra reservada
tk_JOIN	JOIN	Palabra reservada
tk_ON	ON	Palabra reservada
tk_WHERE	WHERE	Palabra reservada
tk_AND	AND	Palabra reservada
tk_OR	OR	Palabra reservada
tk_GROUP	GROUP	Palabra reservada
tk_HAVING	HAVING	Palabra reservada
tk_ORDER	ORDER	Palabra reservada

tk_BY	BY	Palabra reservada
tk_ASC	ASC	Palabra reservada
tk_DESC	DESC	Palabra reservada
tk_LIMIT	LIMIT	Palabra reservada
tk_offset	offset	Palabra reservada
tk_SUM	SUM	Palabra reservada
tk_MIN	MIN	Palabra reservada
tk_MAX	MAX	Palabra reservada
tk_COUNT	COUNT	Palabra reservada
tk_AVG	AVG	Palabra reservada
tk_Azter	*	Símbolo de puntuación
tk_Campo	“biblioteca”	Identificador
tk_LeftParen	(	Símbolo de puntuación
tk_RigthParen	)	Símbolo de puntuación
tk_Punto	.	Símbolo de puntuación
tk_Coma	,	Símbolo de puntuación
tk_SemiColon	;	Símbolo de puntuación
tk_ComillaSimple	‘	Símbolo de puntuación
tk_ComillaDoble	“	Símbolo de puntuación
tk_IntLiteral	52	Literal entero
tk_Literal	lugar	Literal de cadena
tk_OperadorIgualdad	=	Operador
tk_OperadorMayor	>	Operador
tk_OperadorMenor	<	Operador
tk_OperadorMayorIgual	>=	Operador
tk_OperadorMenorIgual	<=	Operador
tk_OperadorDistinto	<>	Operador

Tabla 1 Especificación del lenguaje a interpretar

Las gramáticas constituyen la mejor vía para la descripción sintáctica de los lenguajes. La idea para aplicar una gramática es que se parte de una variable, llamada símbolo inicial, y se aplican repetidamente las reglas gramaticales. El diseño y la definición de la gramática constituye la base para el desarrollo del proceso de interpretación, a continuación se muestra la gramática que define y restringe el lenguaje que será interpretado.

<Query> → <Select> <From> <Where> <GroupBy> <Having> <OrderBy> <Limit>
<Select> → tk_SELECT <CamposSeleccionados>
<CamposSeleccionados> → <CampoSelect> <MasCamposSelect> | tk_Azter
<MasCamposSelect> → tk_coma < CampoSelect > <MasCamposSelect> | e
<From> → tk_FROM <Tabla> <MasTablas>
<MasTablas> → tk_INNER tk_JOIN <Tabla> <CondiciónJoin> <MasTablas> | tk_RIGTH tk_JOIN <Tabla> <CondicionJoin> <MasTablas> | tk_LEFT tk_JOIN <Tabla> <CondiciónJoin> <MasTablas> | tk_FULL tk_JOIN <Tabla> <CondiciónJoin> <MasTablas> | tk_coma <Tabla> <MasTablas> | e
<CondiciónJoin> → tk_ON <Condición> <MasCondiciones>
<Where> → tk_WHERE <Condición> <MasCondiciones> | e
<GroupBy> → tk_GROUP tk_BY <Campo> <MasCampos> | e
<Having> → tk_HAVING <Condición> <MasCondiciones> | e
<OrderBy> → tk_ORDER tk_BY <CampoOrder> <MasCamposOrder> | e
<MasCamposOrder> → tk_coma <CampoOrder> <MasCamposOrder> | e
<Limit> → tk_LIMIT tk_IntLiteral tk_offset tk_IntLiteral | e
<CampoSelect> → tk_LeftParen <Campo> tk_RigthParen <Alias> | <Campo> <Alias> | <CampoOperador> <Alias>
<CampoOperador> → tk_SUM <Campo> | tk_MIN <Campo> | tk_MAX <Campo> | tk_COUNT <Campo> | tk_AVG <Campo>
<Campo> → <Campo_Literal> <Tabla_Campo>
<Alias> → tk_AS <Campo_Literal> | e
<Campo_Literal> → tk_Campo | tk_Literal
<Tabla_Campo> → tk_Punto <Campo_Literal> | e
<Tabla> → <Campo>
<Condición> → <Campo> <OperadorComparación> <CampoDerecho>
<MasCondiciones> → tk_AND <Condición> <MasCondiciones> | tk_OR <Condición> <MasCondiciones> | e
<OperadorComparación> → tk_OperadorIgualdad | tk_OperadorMayor | tk_OperadorMenor | tk_OperadorMayorIgual | tk_OperadorMenorIgual | tk_OperadorDistinto
<CampoDerecho> → <Campo> | tk_ComillaSimple <ValorManual> tk_ComillaSimple
<ValorManual> → tk_IntLiteral | tk_Literal
<MasCampos> → tk_coma <Campo> <MasCampos> | e
<CampoOrder> → <Campo> <Orden>
<MasCamposOrder> → tk_coma <CampoOrder> <MasCamposOrder> | e
<Orden> → tk_ASC | tk_DESC | e

Figura 10 Gramática del lenguaje a interpretar

3.1 Modelo de Implementación

El modelo de implementación describe los componentes a construir y su organización en nodos físicos en los que funcionará el sistema. Está compuesto por los diagramas de despliegue y componentes. Describe como los elementos del diseño se implementan en términos de componentes tales como ficheros de código fuente, ejecutables, librerías y documentos.

3.1.1 Diagrama de componentes

El diagrama de componentes se representa como un grafo de componentes de software unido por medio de las relaciones de dependencia. Entre los componentes se encuentran las clases, interfaces, librerías y componentes ejecutables. Normalmente los diagramas de componentes se utilizan para modelar código fuente, versiones ejecutables, bases de datos físicas y librerías. (21)

A continuación se muestra el diagrama de componentes del CU: Actualizar diseño, el cual está compuesto por tres paquetes principales. Estos paquetes son:

- Paquete de Clases Vista: agrupa los componentes que se encargan de manejar la visualización de la información que recibe del controlador. En este paquete se encuentra la clase **UCQueryBuilder,** que contiene las principales operaciones que se realizan del lado del cliente.

- Paquete de Clases Controlador: contiene las clases que manipulan los eventos del usuario, estas reciben la notificación de la acción solicitada por el mismo, gestionan el evento de entrada, accediendo posteriormente al modelo y otorgando finalmente a los objetos de la vista la tarea de mostrar la interfaz apropiada. En este paquete se encuentra la clase controladora **ParsearConsulta** encargada de recibir la petición de comprobar si la consulta está correctamente estructurada apoyándose en las clases **Lexer**, **Parser** y **Cheker** para dar respuesta a esta petición.

- Paquete de Clases Modelo: agrupa las clases que permiten acceder a la capa de almacenamiento de datos o acceso a datos y responden las instrucciones del controlador. En este paquete se encuentra la clase **Model** que permite obtener los datos del modelo así como las tablas y los campos por tablas. (Ver figura 11)

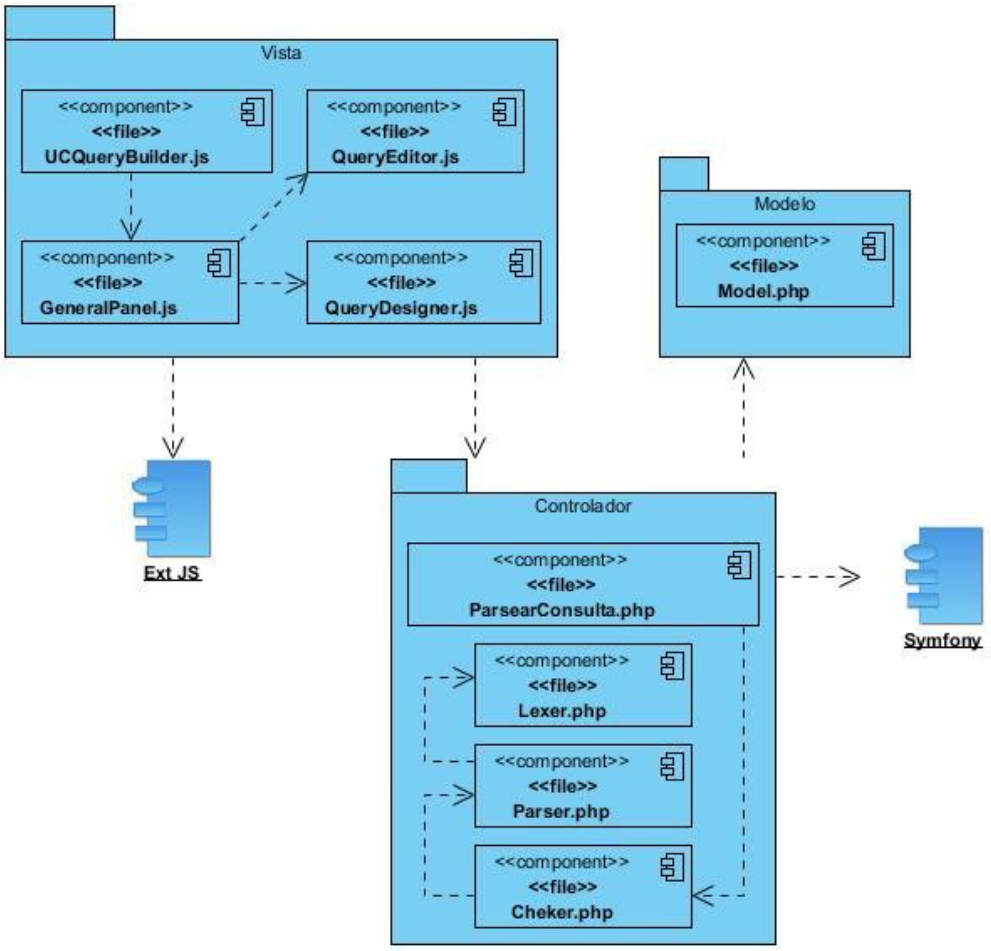

Figura 11 Diagrama de Componentes del CU Actualizar diseño

3.2 Código Fuente

El código fuente es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento. Estas instrucciones son escritas en un lenguaje de programación que consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

3.2.1 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un

código de alta calidad, es de gran importancia para la calidad del software y para obtener un buen rendimiento. (22)

Estilo de Codificación Utilizado:

- Los bloques de código siempre deben estar encerrados por llaves, si consta de una línea no es necesario utilizar llaves.

```
if ($this->lookahead->getKind() != $this->tokenKind->tk_FROM) {
    $this->Campo();
    $this->MasCampos();
} else
    $this->errorReporter[] = "Línea " . $this->lookahead->getLine()
```

Figura 12 Estilo de codificación. Ejemplo 1

- El nombre de métodos y variables, debe permitir que con sólo leerlo se conozca el propósito de la misma.

```
$idModel = $request->getParameter('idModel');
$nameQuery = $request->getParameter('nameQuery');
$codeSQL = $request->getParameter('codeSQL');

public function PalabrasReservadas() {

public function VerificarConsulta() {
```

Figura 13 Estilo de codificación. Ejemplo 2

- Todos los métodos y nombre de clases se escriben en estructura UpperCamelCase, este estilo define el comienzo de cada palabra en mayúscula.

```
class SourceStream {

  public function MasTablas() {
```

Figura 14 Estilo de codificación. Ejemplo 3

- Colocar espacios en blanco entre operadores lógicos-aritméticos y sus operandos.

```
while ($this->c == ' ' || $this->c == '\t' || $this->c == '\n' || $this->c == '\r')
    $this->Consume();
```

Figura 15 Estilo de codificación. Ejemplo 4

- Todas las variables y atributos se escriben en estructura lowerCamelCase, este estilo define la primera letra de cada palabra en mayúscula excepto la primera que la pone en minúscula completa.

```
private $input;
private $lookahead;
private $tokenKind;
private $errorReporter;

$tablasModelo = array();
$bandera = false;
```

Figura 16 Estilo de codificación. Ejemplo 5

Ejemplo de código fuente

A continuación la Figura 17 muestra el fragmento de código del método VerifyQuery() perteneciente a la clase UCQueryBuilder que contiene las principales operaciones que se realizan del lado del cliente. Este método se encarga de obtener la consulta SQL modificada y de verificar si está correctamente estructurada. De ser correcta la sintaxis de la consulta se ejecuta la función UpdateDesign() encargada de actualizar el diseño en su respectiva Vista, esta última funcionalidad también pertenece a la clase antes mencionada.

```
VerifyQuery: function(){
    var idModel = UCQueryBuilderScope.GeneralPanel.getIdModel();
    var nameQuery = UCQueryBuilderScope.GeneralPanel.getQueryName();
    var codeSQL = UCQueryBuilderScope.GeneralPanel.getQueryEditor().getSourceCode();

    Ext.Ajax.request({
        url: '../../report_generator.php/query_builder/ParsearConsulta',
        method: 'POST',
        params: {
            idModel: idModel,
            nameQuery: nameQuery,
            codeSQL: codeSQL
        },
        success: function(response){
            var httpResponse = Ext.decode(response.responseText);
            if(!httpResponse.success){
                UCQueryBuilderScope.GeneralPanel.getQueryEditor().setErrorMessage( httpResponse.sql, httpRespons
                UCQueryBuilderScope.setError(true);
                UCQueryBuilderScope.GeneralPanel.tabPanel.setActiveTab(1);
                UCQueryBuilderScope.GeneralPanel.getQueryEditor().setSourceCode(httpResponse.code);
            }else{
                UCQueryBuilderScope.setError(false);
                UCQueryBuilderScope.GeneralPanel.getQueryEditor().setErrorMessage( "", httpResponse.success);
                UCQueryBuilderScope.GeneralPanel.removeAllEntity();
                UCQueryBuilderScope.GeneralPanel.setIdModel(idModel);

                UCQueryBuilderScope.UpdateDesign(UCQueryBuilderScope.GeneralPanel.getIdModel(), httpResponse.sql
                UCQueryBuilderScope.GeneralPanel.toggleModelIcon();
            }
        },
```

Figura 17 Ejemplo de código fuente

3.3 Pruebas del software

Las pruebas de software constituyen un pilar indispensable para evaluar y determinar la calidad de un software. Son el proceso de ejecución de un programa con la intención de descubrir errores previos a la entrega al usuario final. Estas pruebas se pueden describir como una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, los resultados son observados y registrados, haciéndose una evaluación de diferentes aspectos del sistema o componente.

3.3.1 Niveles de Prueba

A la hora de evaluar un sistema se debe comenzar por los componentes más simples y pequeños e ir avanzando hasta probar todo el software en su conjunto. Las pruebas se aplican durante todo el ciclo de desarrollo del software para diferentes objetivos y en distintos niveles de trabajo, de los cuales se seleccionaron: pruebas a nivel de desarrollador e integración para comprobar el correcto funcionamiento del sistema en todos los escenarios. Para ello es necesario identificar qué técnicas, tipos y métodos de pruebas pueden ser adecuados para aplicar en este momento de revisión del software. A continuación se describen los utilizados durante el proceso de pruebas al sistema.

Tipos de Prueba

Pruebas funcionales: es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software.

Pruebas de integración: son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias.

Método de Prueba

Caja Negra: las pruebas de Caja Negra se centran en los requerimientos funcionales del software. Permite al ingeniero de software derivar conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales de un programa. Para llevar a cabo este método se seleccionó la técnica de Partición de Equivalencia que divide el campo de entrada en clases de datos: válidos e inválidos, y se prueba el funcionamiento del sistema frente a estas clases de datos.

3.3.2 Diseño de Caso de Prueba

El caso de prueba específica la forma de probar un sistema incluyendo las entradas, salidas y resultados esperados, así como bajo qué condiciones debe probarse el sistema. Su tarea principal es verificar el cumplimiento de un objetivo en particular.

Para realizar las pruebas se utilizaron juegos de datos válidos e inválidos haciendo uso de la técnica de partición de equivalencia. A continuación se muestra el caso de prueba asociado al caso de uso Actualizar diseño.

Escenario	Variables	Descripción	Respuesta del sistema	Flujo Central
EC 1.1 Modificar Consulta con datos correctos	N/A	En este escenario se realiza la modificación de la consulta contenida en la Vista SQL correctamente.	El sistema verifica que los datos de la consulta modificada estén correctamente y actualiza la Vista del diseño.	**1.** Módulo Diseñador de Consultas/ Vista SQL **2.** Se introduce los datos de la consulta correctamente. **3.** Se selecciona la Vista de diseño. **4.** El sistema verifica que la consulta modificada este bien estructurada. **5.** El sistema actualiza la Vista de diseño.
EC 1.2 Modificar consulta con datos incorrectos	N/A	En este escenario se realiza la modificación de la consulta contenida en la Vista SQL con datos incorrectos.	El sistema verifica que los datos de la consulta modificada son incorrectos y le muestra al usuario el tipo de error que se identificó.	**1.** Módulo Diseñador de Consultas/ Vista SQL **2.** Se introduce los datos de la consulta incorrectamente. **3.** Se selecciona la Vista de diseño. **4.** El sistema verifica que la consulta modificada este bien estructurada y detecta los errores en la misma. **5.** El sistema no cambia de Vista y le muestra al usuario el tipo de error existente en la consulta.

Tabla 2 Caso de prueba del CU Actualizar diseño

Este proceso permitió verificar el cumplimiento de los requisitos funcionales del sistema, donde los resultados de las pruebas que no fueron satisfactorios pasaron a ser no conformidades y se resolvieron posteriormente.

3.4 Resultados de las pruebas

Caja Negra

Después de realizar las pruebas de caja negra mediante los casos de prueba asociados a cada caso de uso, se comprobó el correcto funcionamiento del intérprete SQL y se verificó la validez de los datos que se modifican en la consulta. Se detectaron 4 no conformidades en el desarrollo del intérprete SQL a las cuales se les dio tratamiento y fueron resueltas en la tercera iteración de la fase de pruebas.

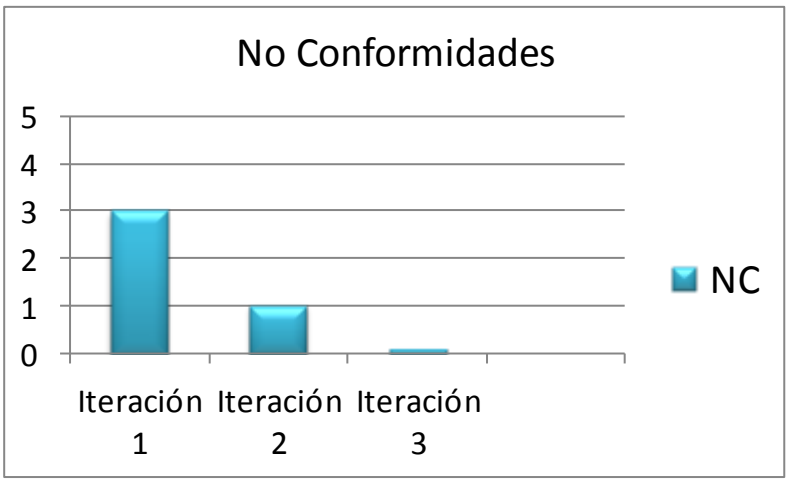

Figura 18 Gráfica de No Conformidades

Integración

Una vez terminado el intérprete SQL se integra al módulo Diseñador de Consultas hasta que funciona como un todo. Se prueban las funcionalidades del módulo para asegurar que los datos que se introducen en la consulta modificada correspondan con los necesarios para el correcto funcionamiento del mismo. Se comprueba que queden validados todos los datos que se introducen en la vista SQL para modificar la consulta que se encuentra en diseño.

Pasos para realizar la integración:

1. Se agregó la clase controladora ParsearConsulta en el paquete de clases controladora del módulo, para recibir y dar respuesta a la petición de verificar si la consulta modificada esta correctamente estructurada.
2. Se agregó al paquete de librerías del GDR, el paquete ManipulateQuery que contiene las clases del intérprete SQL.
3. Para garantizar la carga dinámica de las clases del intérprete se debe actualizar el fichero autoload.yml.php.
4. Se agregaron las funciones VerifyQuery() y UpdateDesign() a la clase UCQueryBuilder. El primer método se encarga de obtener la consulta SQL modificada y de verificar si está correctamente estructurada. De ser correcta la sintaxis de la consulta se ejecuta la función UpdateDesign() encargada de actualizar el diseño en su respectiva Vista.

Luego de realizar las pruebas de caja negra se integró el intérprete SQL al módulo Diseñador de Consultas y se pudo comprobar que está debidamente integrado y que funciona correctamente. Cuando se modifican los datos en la vista SQL de la consulta que se encuentra en diseño y se cambia de vista, se actualiza la vista del diseño de la consulta modificada.

3.4 Conclusiones del capítulo

Con la realización de este capítulo, se definió la gramática que guió el proceso de interpretación, se realizó la especificación del lenguaje a interpretar. Se describió cómo fue implementada la solución en términos de componentes de cada uno de los casos de uso del sistema y se mostraron sus relaciones en el diagrama de componentes. Se reflejaron los estándares de codificación que sirvieron para lograr una implementación organizada y un código de alta calidad. Se realizaron pruebas de integración y funcionales que validaron la correcta implementación de los requisitos funcionales y demostraron el correcto funcionamiento de intérprete SQL. Se detectaron 4 no conformidades que fueron resueltas durante la fase de pruebas.

Conclusiones Generales

Una vez culminado el trabajo de diploma se considera que se cumplió el objetivo general y las tareas trazadas alcanzando los siguientes resultados:

❖ Se desarrolló un estudio previo sobre los Intérpretes y el lenguaje SQL que posibilitó obtener los conocimientos necesarios para dar comienzo al diseño e implementación del Intérprete SQL.

❖ Con la correspondiente modelación del dominio, el análisis de los requisitos y la especificación de los casos de uso del sistema se sentaron las bases para desarrollar el Intérprete SQL, documentándose todo el proceso de desarrollo obteniéndose los artefactos correspondientes.

❖ Partiendo del resultado del diseño, se realizó la implementación del intérprete, para ello se modeló el sistema en términos de componentes, se dieron a conocer las técnicas usadas en la solución del problema, se definió la gramática que guió el proceso de interpretación y se mostraron ejemplos de las implementaciones más relevantes.

❖ Para garantizar la calidad del Intérprete SQL desarrollado se realizaron pruebas en dos niveles, en el nivel de desarrollador y de integración, obteniéndose un producto libre de errores y listo para su uso.

❖ Se realizó un Intérprete SQL que facilita los recursos necesarios para la creación de reportes a partir de consultas prediseñadas.

Recomendaciones

Luego de haber cumplido los objetivos propuestos mediante la realización del trabajo, se recomienda:

- Implementar el soporte de consultas anidadas.
- Implementar el soporte para procedimientos almacenados.

BIBLIOGRAFÍA

1. **Bakken, Stig Sæther.** php.uci.cu. [En línea] [Citado el: 9 de 11 de 2012.] http://php.uci.cu/search.php.
2. **Blank, Isabel, Herrera, Larissa and Ortiz, Miguel. Pruebas** de Funcionalidad. mayo de 2005. http://carolina.terna.net/ingsw3/datos/Pruebas_Funcionales.pdf.
3. **Boutelle, Jon, Ranjan, Amit y Sinha, Rashmi. SlideShare.** [En línea] 2006. [Citado el: 22 de 11 de 2012.] http://es.slideshare.net/Anita325/lenguajes-de-bases-de-datos-10238259.
4. **Caudillo León, Juan Raúl.** Redes Educación Sistemas. [En línea] http://conocimientoysistemas.wordpress.com/tag/caracteristicas-xml/.
5. **Craig Larman**. UML y Patrones. [En línea] *www.publidisa.com/preview-libro-9788483229279.pdf*.
6. **Dorado, Andrés.** Patrones de diseño. [En línea] http://www.slideshare.net/2008PA2Info3/tema-de-revision-2003
7. EcuRed. [En línea] [Citado el: 14 de 11 de 2012.] http://www.ecured.cu/index.php/OpenUp .
8. EcuRed. [En línea] http://www.ecured.cu/index.php/SQL.
9. Fabien Potencier. Symfony la Guía Definitiva.
10. Figueroa, Roberth, J. SOLÍS, Camilo y A. Cabrera, Armando. Metodologías tradicionales vs. Metodologías ágiles. Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación : s.n., 2008.
11. **IAN Somerville**. Ingeniería del Software. [En línea]http://www.filecrop.com/ian-sommerville-ingenieria-de-software-septima-edicion.pdf.html.
12. **Liesner Acevedo Martínez**. Técnicas de Compilación: Manual Práctico para estudiantes de Informática.
13. **Module for Hosting.** [En línea] http://www.modulehosting.com/apache.html.
14. **Monne Roque, Diana y Lores Sánchez, Linet.** *Aplicación de las pruebas de liberación al Sistema Informático de Genética Médica.* La Habana : s.n., 2009.
15. netbeans.org. [En línea] [Citado el: 20 de 11 de 2012.] http://netbeans.org/community/releases/72/relnotes.html.
16. **PATRONES DE DISEÑO Y FRAMEWORKS.** [En línea] [Citado el: 21 de febrero de 2013.] http://ingenieriasw2.blogspot.com/p/patrones-de-diseno-y-frameworks.html.

17. **Pérez, Javier Eguíluz.** Manual de java script. [En línea] www.librosweb.es.
18. php.net. [En línea] [Citado el: 12 de 11 de 2012.] http://php.net/releases/5_3_0.php.
19. **Reynoso, C. B. (2004).** Introducción a la Arquitectura de Software. Recuperado el 28 de febrero de 2011,de http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp.
20. **Slideshare**. [En línea] [Citado el: 2 de abril de 2013.] http://www.slideshare.net/joshell/diagramas-uml-componentes-y-despliegue.
21. **Slideshare.** Slideshare. [En línea] [Citado el: 1 de 6 de 2011.] http://www.slideshare.net/dwebslide/estandares-de-diseo-web.
22. **Steve Burbeck**. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). [En línea] [Citado el: 24 de 1 de 2011.] http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.
23. **Tello, Jesús Cáceres.** *Diagramas de Casos de Uso*. Universidad de Alcalá Departamento de Ciencias de Computación.
24. **Universidad de las Ciencias Informáticas.** Especificación de Requisitos de Software V2.0 del Generador Dinámico de Reportes. s.l. : Sistema de Información de Gobierno.
25. **Universidad Jaime.** *Ingeniería Informática. Procesadores de lenguajes. Estructura de los compiladores e interprétes.* Castellón de la Plana (España) : s.n., 2009.
26. **Visual Paradigm.** Visual Paradigm. [En línea] http://www.visual-paradigm.com/product/vpuml/.

Referencias Bibliográficas

1. **Solorzano, Hector Valencia.** Todo de programacion. [En línea] [Citado el: 12 de 11 de 2012.] http://www.todo-programacion.com.ar/archives/2005/04/ Intérpretes_y_c.html.

2. EcuRed. [En línea] http://www.ecured.cu/index.php/Intérprete_(Informática).

3. **Liesner Acevedo Martínez**. Técnicas de Compilación: Manual Práctico para estudiantes de Informática.

4. **I, Universidad Jaime.** *Ingeniería Informática. Procesadores de lenguajes. Estructura de los compiladores e interprétes.* Castellón de la Plana (España) : s.n., 2009.

5. EcuRed. [En línea] http://www.ecured.cu/index.php/SQL.

6. **Figueroa, Roberth, J. SOLÍS, Camilo y A. Cabrera, Armando.** *Metodologías tradicionales vs. Metodologías ágiles.* Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación : s.n., 2008.

7. **Pérez, Fabián Bermeo.** Metodología RUP. [En línea] 15 de 11 de 2012. http://fabianbermeop.blogspot.com/2010/12/metodologia-rup-desarrollo-de-software.html.

8. **Figueroa, Roberth, J. SOLÍS, Camilo y A. Cabrera, Armando.** *Metodologías tradicionales vs. Metodologías ágiles.* Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación : s.n., 2008.

9. EcuRed. [En línea] [Citado el: 14 de 11 de 2012.] http://www.ecured.cu/index.php/OpenUp .

10. **Patricia López, Francisco Ruiz.** Lenguaje Unificado de Modelado - UML. [En línea] http://ocw.unican.es/ensenanzas-tecnicas/ingenieria.../is1-t02-trans.pdf.

11. EcuRed. [En línea] [Citado el: 17 de 11 de 2012.] http://www.ecured.cu/index.php/Visual_Paradigm.

12. Open Courses Atlantic International University [En línea] [Citado el: 17 de 11 de 2012.] http://cursos.aiu.edu/Programacion%20de%20Computadoras/PDF/Tema%201.pdf

13. php.net. [En línea] [Citado el: 12 de 11 de 2012.] http://php.net/releases/5_3_0.php

14. **Caudillo León, Juan Raúl.** Redes Educación Sistemas. [En línea] http://conocimientoysistemas.wordpress.com/tag/caracteristicas-xml/.

15. EcuRed. [En línea] [Citado el: 15 de 11 de 2012.] http://www.ecured.cu/index.php/EXtreme_Programming..

16. netbeans.org. [En línea] [Citado el: 20 de 11 de 2012.] http://netbeans.org/community/releases/72/relnotes.html.

17. **Craig Larman**. UML y Patrones . [En línea] *www.publidisa.com/preview-libro-9788483229279.pdf*

18. **Olivares, Jose Rolando Lafaurie.** Sistema para la generacion de reportes en la plataforma alasGRATO: Desarrollo del módulo Reportador. : s.n., 2008.

19. **Steve Burbeck**. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). [En línea] [Citado el: 24 de 1 de 2011.] http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

20. **Dorado, Andrés.** Patrones de diseño. [En línea] http://www.slideshare.net/2008PA2Info3/tema-de-revision-2003

21. **IAN Somerville**. Ingenieria del Software . [En línea]http://www.filecrop.com/ian-sommerville-ingenieria-de-software-septima-edicion.pdf.html

22. **Slideshare.** Slideshare. [En línea] [Citado el: 1 de 6 de 2011.] http://www.slideshare.net/dwebslide/estandares-de-diseo-web.

UCI
Universidad de las Ciencias Informáticas

Centro de Tecnologías de Gestión de Datos
DATEC

La Habana, 14 de mayo del 2013
"Año 54 de la Revolución".

ACTA DE ACEPTACIÓN

De una parte, el Centro de Tecnologías de Gestión de Datos, en lo sucesivo DATEC, de la Universidad de las Ciencias Informáticas, representado en este acto por: Ing. Aldis Joan Abreu Saladeña, y de **otra parte** el estudiante: Jorge Antonio Bernal Rojas.

Primero: Que en cumplimiento de los requisitos funcionales han sido efectuadas las implementaciones correspondientes.

CONSIDERANDO: Que los hitos realizados han sido desarrollados con la calidad requerida y bajo las condiciones pactadas y aprobadas por **Las Partes**.

CONSIDERANDO: Que los hitos que se han ejecutado cumplen con los requerimientos establecidos.

POR TANTO: Las Partes acuerdan formalizar mediante la presente Acta, la aceptación del producto: Intérprete SQL para el módulo Diseñador de consultas del GDB

Y para que así conste, se extiende la presenta **Acta** en dos (2) ejemplares, rubricados por **Las Partes**.

Universidad de las Ciencias Informáticas
Decanato Facultad

Jorge Antonio Bernal Rojas
Entrega

Ing. Aldis Joan Abreu Saladeña
Recibe

Figura 19 Acta de Aceptación

GLOSARIO DE TÉRMINOS

CASE: Ingeniería de Software Asistida por Computación.

CU: Caso de Uso.

CUS: Caso de Uso del Sistema.

DATEC: Centro de Tecnologías de Gestión de Datos.

DCL: Data Control Language (Lenguaje de Control de Datos).

DDL: Data Definition Language (Lenguaje de Definición de Datos).

DML: Data Manipulation Language (Lenguaje de Manipulación de Datos).

GDR: Generador Dinámico de Reportes.

GoF: Gang of Four.

GRASP: General Responsability Assignment Software Patterns (Patrones de Software para la asignación General de Responsabilidades).

IDE: Entorno desarrollo integrado.

Interoperabilidad: Habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.

MIT: Es una licencia de software libre originada en el Instituto de Tecnología de Massachusetts (MIT) que permite modificar el código fuente del software.

OpenUP: Proceso Unificado Abierto.

RUP: Proceso Unificado de Rational.

SQL: Structured Query Language (Lenguaje de Consulta Estructurado).

UML: Unified modeling language (Lenguaje Unificado de Modelado).

XML: Lenguaje de Marcas Extensible.

XP: eXtreme Programming (Programación Extrema).