UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 6

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

"Componente para la abstracción de la capa de acceso a datos del sistema GeolMin"

Autora:
Milenis Fernández Díaz

Tutor:
Ing. Dagoberto Antonio Suárez Morales

La Habana, Junio de 2011
"Año 53 de la Revolución"

Dedicatoria

Dedico este trabajo a mis padres Amparo y Félix,
A mi querida abuela Rosa, a mis tías Silvia y Yamilet,
A mi hermano Miguel Ángel,
A mi novio José Gabriel.

Agradecimientos

Agradezco especialmente a mis padres, tíos y abuelos.
A mi novio José Gabriel por su apoyo incondicional.
A mi tutor, el Ing. Dagoberto Antonio Suárez.
A los profesores Eddy Dangel Quezada y
Yunier Alexander Pimienta.
A mi amiga Lisbet.

Declaración de autoría

Declaro que soy la única autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Milenis Fernández Díaz	Ing. Dagoberto A. Suárez Morales
_____	_____
Firma del autor	**Firma del tutor**

Resumen

Las empresas mineras en el ámbito nacional generan grandes volúmenes de información, como resultado de las actividades de prospección y exploración de los recursos mineros. Con vistas a apoyar estas actividades, en el departamento Geoinformática del centro de desarrollo de software GEySED se construye el sistema GeolMin. Este sistema necesita un mecanismo que le permita manipular la información con independencia de la tecnología de base de datos utilizada.

El presente trabajo de diploma tiene como resultado un componente de abstracción de las bases de datos del sistema GeolMin (CABD 1.0). Dicho componente proporciona un conjunto de clases que permiten manipular la información contenida en una base de datos. Además, permite a los programadores utilizar una sintaxis única e independiente del sistema gestor para el acceso a los datos.

El componente CABD fue elaborado siguiendo los pasos propuestos por la metodología FDD, empleando el lenguaje de modelado UML y la herramienta CASE Visual Paradigm. Se utilizó C++ como lenguaje de programación y Qt Creator como entorno de desarrollo. En su primera versión se encuentra disponible para plataformas Linux y brinda soporte a los sistemas gestores SQLite y PostgreSQL, siendo fácilmente extensible a otros gestores.

Palabras claves: abstracción de bases de datos, capa de abstracción, componente, manipulación de información.

Contenido

Índice de tablas

Índice de figuras

Introducción

En la actualidad, la mayoría de las empresas emplean las Tecnologías de la Información (TI) para automatizar sus procesos en aras de reducir los costos, lograr eficiencia y ofrecer mejores servicios a los clientes. La automatización de estos procesos también facilita la toma de decisiones al proporcionar la información necesaria de forma rápida y segura. Además, la integración de las tecnologías y los conocimientos favorece la obtención de resultados en plazos razonables, contribuyendo de manera significativa al mejoramiento de las actividades empresariales.

Las empresas mineras en el ámbito nacional, representadas por la Unión Geólogo-Minera y la Unión del Níquel, han optado por beneficiarse del uso de las TI a través de la informatización de algunas de sus actividades. Estas empresas motivadas por el deseo de mejorar constantemente sus procesos, muestran interés en integrar los datos recopilados en distintas etapas de sus operaciones, en herramientas computacionales que puedan traducirlos en información valiosa para sus negocios. El uso de sistemas informáticos en el campo de la minería permite una mejor gestión del proceso minero, proporcionando los datos necesarios para la toma de decisiones.

En el departamento Geoinformática del centro de desarrollo de software GEySED, perteneciente a la Universidad de las Ciencias Informáticas (UCI), existe un proyecto llamado Minería en el cual se crean soluciones para la gestión de datos geólogos-mineros. En este proyecto actualmente se construye el sistema GeolMin como herramienta de apoyo en las actividades de prospección y exploración de los recursos mineros.

Como resultado de la informatización de estas actividades, se generarán grandes volúmenes de información y con vistas a garantizar su persistencia y manipulación, las empresas mineras emplearán tecnologías de bases de datos (BD). La sustitución de una tecnología de BD por otra hace necesario implementar las funcionalidades requeridas para establecer la conexión con el nuevo tipo de BD, así como sobrescribir las consultas anteriormente definidas.

Partiendo de esta situación problemática se determina como **problema científico**: ¿Cómo abstraer el acceso a los datos de las BD utilizadas en GeolMin?

En la presente investigación se plantea como **objeto de estudio**: La manipulación de la información de una BD. Dentro de dicho objeto de estudio se enmarca el siguiente **campo de acción**: La capa de abstracción de las BD del producto GeolMin.

Para resolver el problema se plantea como **objetivo general** de la investigación: Desarrollar un componente para la abstracción de las BD del sistema GeolMin.

Para dar cumplimiento al objetivo general se han trazado las siguientes **tareas de la investigación**:

1. Caracterizar las funcionalidades de la abstracción de BD.
2. Caracterizar las tendencias y tecnologías actuales.
3. Elaborar el modelo global del componente.
4. Elaborar la lista de rasgos del componente.
5. Elaborar el cronograma de diseño e implementación.
6. Diseñar los rasgos definidos.
7. Implementar los rasgos definidos.
8. Validar la solución mediante pruebas.

Para el desarrollo de la investigación se emplearon los siguientes **métodos científicos**:

Métodos teóricos:

- Analítico-sintético: Se utilizó en el análisis de los elementos relacionados con la abstracción de BD, así como las tendencias y tecnologías actuales; y posteriormente en la síntesis de los resultados obtenidos.
- Histórico-lógico: Se empleó en el análisis de las tendencias y tecnologías actuales empleadas en la abstracción de BD, y en el análisis de otras soluciones existentes.
- Modelación: Se utilizó durante la construcción del modelo global y en la modelación de los diagramas correspondientes al diseño del componente.

Métodos empíricos:

- Experimento: Se utilizó para detectar las deficiencias presentes en el componente, así como verificar su correcto funcionamiento.

El análisis de los elementos anteriormente expuestos conduce a plantear como **idea a defender**: El componente para la abstracción de las BD de GeolMin permitirá manipular la información de la aplicación con independencia del tipo de BD utilizada.

Se espera como **posible resultado** obtener un componente de abstracción de las BD del sistema GeolMin, que garantice la independencia de la aplicación con respecto al tipo de BD utilizada.

El presente trabajo está estructurado en 5 capítulos:

Capítulo 1. Conceptos y antecedentes relacionados con el componente: Son expuestos los principales elementos teóricos necesarios para el desarrollo de la investigación. Incluye el estado del arte a nivel internacional, nacional y de la universidad.

Capítulo 2. Tendencias y tecnologías: Se caracterizan las tendencias y tecnologías a utilizar en el desarrollo de la solución informática justificando su selección.

Capítulo 3. Características del componente: Se construye un modelo global del componente con vistas a lograr un entendimiento del mismo. Se identifican las características que debe tener y se diseña un cronograma para su posterior diseño e implementación.

Capítulo 4. Diseño del componente: Se define la arquitectura de software y los patrones de diseño a utilizar. Se modela el componente de abstracción de BD a través de un diagrama de clases.

Capítulo 5. Construcción y pruebas: Se define el estándar de codificación a utilizar, así como las pruebas a las que será sometida la solución informática.

Capítulo 1. Conceptos y antecedentes relacionados con el componente

1.1 Introducción

En este capítulo se enuncian los conceptos fundamentales asociados al dominio del problema, con vistas a posibilitar un mejor entendimiento del mismo. También se describe el dominio del problema, la situación problemática y el objeto de estudio de la investigación. Finalmente, se analizan otras soluciones existentes que de una forma u otra brindan respuesta al problema planteado.

1.2 Conceptos asociados al dominio del problema

Datos e información

Los términos datos e información en ocasiones se usan indistintamente, mientras que en otras los autores prefieren distinguir las diferencias entre ambos términos. Estos últimos se refieren a los datos como lo que realmente se almacena en la BD, y a la información como el significado que estos adquieren para un usuario (Date 2003).

Los datos constituyen hechos sobre los cuales se fundamentan las necesidades de información y procesamiento de determinada organización. A partir de los mismos se pueden inferir nuevos hechos. Por sí solos no tienen valor, deben ser interpretados para que se conviertan en información útil.

La información constituye un recurso valioso para las organizaciones al impactar de manera positiva la planificación y la toma de decisiones. La obtención de información certera y precisa en el menor tiempo posible, constituye el objetivo de cualquier sistema o programa informático (Ezequiel Rozic 2004).

Base de Datos (BD)

Las BD constituyen una tecnología informática ampliamente utilizada por las empresas e instituciones para satisfacer sus necesidades de información. Según Rosa María Matos, una BD es un "conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora" (Mato García 1999). C.J. Date ofrece una definición similar en la cual se refiere a una BD como "un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada"

(Date 2003). A los datos contenidos en una BD les llama persistentes, ya que una vez que son almacenados solo pueden ser actualizados o eliminados por alguna solicitud explícita al Sistema Gestor de Base de Datos (SGBD).

Una definición más abarcadora fue dada por Adoración de Miguel Castaño, Mario Pianttini y Esperanza Marcos Martínez. Ellos definieron una BD como "una colección o depósito de datos integrados, almacenados en soporte secundario y con redundancia controlada". Expresan también que los datos compartidos por diferentes usuarios y aplicaciones deben mantenerse independientes de estos. Su definición se debe apoyar en un modelo de datos que capte las interrelaciones y restricciones existentes en el mundo real (De Miguel Castaño, Piattini Velthuis et al. 1999).

Una BD también se define como "un conjunto de información estructurada en registros y almacenada en un soporte electrónico legible desde un ordenador. Cada registro constituye una unidad autónoma de información que puede estar a su vez estructurada en diferentes campos o tipos de datos que se recogen en dicha BD" (Yunta 2001). Según Noel L. Núñez, una BD es "un conjunto exhaustivo de datos estructurados, fiables y homogéneos, organizados independientemente de su utilización e implementación en una computadora, accesibles en tiempo real, que pueden compartir varios usuarios con necesidades de información diferentes y no predecibles en el tiempo" (Camallea 2004).

Existen distintos modelos de organización de las BD, los cuales son: de forma jerárquica, en red, relacionales y orientados a objetos. El modelo relacional es uno de los más populares. Una BD relacional se define como un conjunto de relaciones (representadas en tablas) que almacenan una colección específica de datos estructurados. Cada tabla puede variar con el transcurso del tiempo y se identifica de manera única por medio de un nombre (Ezequiel Rozic 2004).

Sistema Gestor de Base de Datos (SGBD)

Adoración de Miguel Castaño, Mario Pianttini y Esperanza Marcos en su libro "Diseño de Bases de Datos Relacionales" expresan que un SGBD constituye "un conjunto coordinado de programas, procedimientos, lenguajes, herramientas, que suministra, tanto a los usuarios no informáticos como a los analistas, programadores o administradores de una BD, los medios necesarios para describir y manipular los datos

integrados en la BD, manteniendo su integridad, confidencialidad y disponibilidad" (De Miguel Castaño, Piattini Velthuis et al. 1999).

Rosa María Matos define un SGBD como "el software que permite la utilización y/o la actualización de los datos almacenados en una (o varias) BD por uno o varios usuarios desde diferentes puntos de vista y a la vez" (Mato García 1999). Según Noel L. Núñez un SGBD es un "conjunto de programas de propósito general, que permite a los usuarios controlar el acceso y la utilización de la base de datos, para incluir, modificar o recuperar información, incluyendo prestaciones con el fin de conseguir la independencia, integridad y seguridad de los datos, y la concurrencia de los usuarios" (Camallea 2004).

Un SGBD constituye un conjunto de programas, herramientas y lenguajes que permiten la definición y manipulación de los datos almacenados en una BD. Estos sistemas garantizan la independencia de los datos con respecto a las aplicaciones, así como la seguridad de los mismos. De esta forma es posible modificarlos sin realizar cambios en las aplicaciones.

Entre sus principales funcionalidades se encuentran: la creación y mantenimiento de BD; garantizar el acceso, definición, actualización y recuperación de los datos; así como permitir el control centralizado de estos. También permiten garantizar la integridad, seguridad y fácil manipulación de los datos; y evitar las redundancias e inconsistencias de los mismos (Mato García 1999).

Figura 1. Representación de un SGBD.

Los SGBD actúan como interfaz entre las BD físicas, los programas de aplicación y los usuarios del sistema (Figura 1). Ocultan a los usuarios detalles acerca del almacenamiento físico de los datos a través de los siguientes niveles de abstracción (Date 2003) (Figura 2):

- Nivel interno o físico: Nivel más bajo de abstracción el cual describe en detalle cómo se almacenan realmente los datos.
- Nivel lógico o conceptual: Describe qué datos se almacenan en la BD y qué relaciones existen entre estos. Generalmente los programadores y administradores de BD trabajan en este nivel.
- Nivel externo o de vista: Nivel más alto de abstracción y más próximo a los usuarios. Está relacionado con la forma en que los usuarios individuales ven los datos.

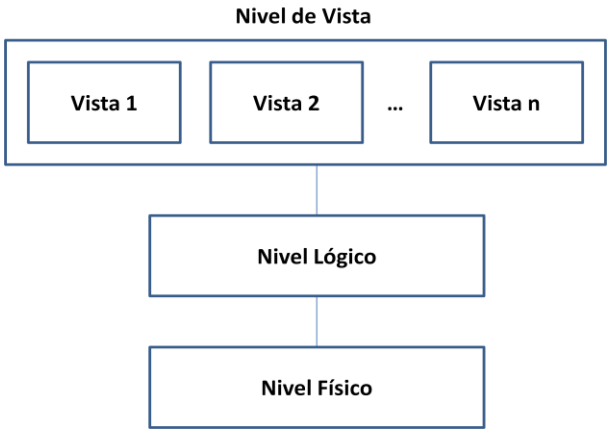

Figura 2. Niveles de abstracción de los SGBD.

Abstracción

Según el Diccionario de la Real Academia Española (DRAE), la asbtracción es la acción y efecto de abstraer. Consiste en "separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción" (RAE 2011). De acuerdo a la enciclopedia británica Merriam Webster, el término abstracción se refiere a "un concepto o una idea no asociada a cualquier caso específico" (Merriam_Webster 2011).

La abstracción es un proceso mental mediante el cual se ignoran algunas características de los objetos, centrando la atención en las cualidades esenciales. De esta forma es posible tratar a objetos diferentes como si fueran la misma cosa. También se facilita la comprensión de sistemas complejos que contienen numerosos detalles y relaciones, reduciendo la cantidad de información a tener en cuenta.

1.3 Descripción actual del dominio del problema

La UCI cuenta con varios centros de desarrollo de software, entre los cuales se encuentra GEySED. Este centro está compuesto por los departamentos: Geoinformática y Señales Digitales, encargados de desarrollar aplicaciones informáticas en el campo de la Geoinformática y el procesamiento de señales digitales, respectivamente. El departamento de Geoinformática cuenta con un proyecto llamado Minería, en el cual se desarrollan soluciones para la gestión de los datos geólogos-mineros.

En este proyecto actualmente se construye el software GeolMin utilizando C++ como lenguaje de programación. Este producto se encuentra dirigido a las empresas del sector minero en el ámbito nacional, las cuales están representadas por la Unión Geólogo-Minera y la Unión del Níquel. La Unión Geólogo-Minera es la entidad estatal cubana encargada de la prospección, exploración y explotación de minerales metálicos y no metálicos, excepto el níquel. La Unión del Níquel es la entidad estatal cubana encargada de la explotación y procesamiento del níquel y el cobalto.

El software GeolMin servirá como herramienta de apoyo en las actividades de prospección y exploración de los recursos mineros. Permitirá la modelación y visualización de yacimientos minerales, la estimación de la cantidad de recursos mineros y reservas de minerales, el diseño y planificación de la mina, entre otras funcionalidades. Como resultado de la informatización de estas actividades, se generarán grandes volúmenes de información y con vistas a garantizar su persistencia y manipulación, las empresas mineras emplearán tecnologías de BD.

La implementación del acceso a los datos del sistema GeolMin para un SGBD específico establece una dependencia de la aplicación con respecto al tipo de BD utilizada; dificultando los posibles cambios de estas tecnologías que puedan ocurrir. Si en el futuro se decide sustituir una tecnología de BD por otra, será necesario implementar las funcionalidades requeridas para establecer la conexión con el nuevo tipo

de BD, así como sobrescribir las consultas anteriormente definidas. Brindar soporte a la nueva tecnología conllevará dedicar más tiempo y esfuerzo por parte del equipo de desarrolladores.

La programación del acceso a los datos constituye un proceso complejo, debido a que los desarrolladores deberán dominar los lenguajes ofrecidos por los distintos SGBD para manipular la información; teniendo en cuenta que las características de estos lenguajes varían de un gestor a otro. Los desarrolladores también deberán tener presente otros detalles de las BD, como por ejemplo las tablas y las filas, los cuales complejizan la implementación del acceso a los datos.

1.4 Manipulación de información de una base de datos

La manipulación de información en una BD constituye uno de los aspectos más importantes para cualquier sistema informático que emplee dicha tecnología. Comprende diferentes actividades como el almacenamiento, la recuperación, la actualización y la eliminación de los datos. Es posible realizar estas acciones mediante las facilidades que brindan los SGBD. Estos sistemas poseen lenguajes especiales llamados sublenguajes de datos (DSL, *Data Sublanguage*) que permiten manipular la información.

Los sublenguajes de datos se emplean para "tratar los objetos de la BD y sus operaciones". Están compuestos por al menos dos lenguajes subordinados: "un lenguaje de definición de datos (DDL, *Data Definition Language*), el cual garantiza la definición o descripción de los objetos de la BD, y un lenguaje de manipulación de datos (DML, *Data Manipulation Language*), que garantiza la manipulación o tratamiento de esos objetos" (Mato García 1999). Algunos gestores incluyen un lenguaje de control de los datos (DCL, *Data Control Language*) que permite controlar los permisos de los objetos de la BD.

SQL (Lenguaje Estructurado de Consultas) es uno de los sublenguajes de datos más populares en los sistemas relacionales. Para la definición de los datos emplea las instrucciones: CREATE (usado para crear nuevas tablas e índices), ALTER (modificar tablas agregando o eliminando campos) y DROP (eliminar tablas e índices). Para la manipulación de los datos emplea las instrucciones: SELECT (definir consultas a los datos), INSERT (insertar registros en una tabla), UPDATE (actualizar los registros de una tabla) y DELETE (eliminar registros de una tabla) (Ezequiel Rozic 2004).

Los sublenguajes de datos son aprovechados por las aplicaciones o sistemas informáticos para operar la información almacenada, llevando a cabo tareas específicas. Con este fin se implementan un conjunto de clases y funciones que conforman lo que se conoce como capa de acceso a los datos. Esta capa es la encargada de proveer a los programas de aplicación el acceso a los datos almacenados de forma persistente.

El empleo de distintas tecnologías de BD y las posibles sustituciones de una tecnología por otra, hacen necesario que las soluciones informáticas brinden soporte para distintos tipos de BD. Por estas razones se recomienda la implementación de una capa de abstracción que permita manipular la información con independencia del tipo de BD utilizada. La capa de abstracción de BD es una Interfaz de Programación de Aplicaciones (API, *Applications Programming Interface*) que permite la comunicación entre una aplicación informática y diferentes SGBD, abstrayendo las diferencias existentes entre estos sistemas (Otto and Hinderink 2004).

Su utilización ofrece las siguientes ventajas:

- Reduce la complejidad: Libera a los desarrolladores de tratar con el mecanismo de persistencia de datos. Los desarrolladores ya no tendrán que preocuparse acerca de la procedencia de los datos, ni de la forma en que estos son almacenados; excepto cuando se desee lograr un alto rendimiento. (Nguèn 2008).
- Portabilidad: Garantiza la independencia de las aplicaciones respecto a un tipo específico de tecnología de BD, facilitando la sustitución de una tecnología por otra sin tener que implementar o sobrescribir las consultas (Army_Net_Centric_Data 2009).
- Homogeneidad: Permite a los programadores utilizar una misma sintaxis para el acceso a los datos, independientemente del SGBD empleado (Army_Net_Centric_Data 2009).
- Reutilización: Constituye un componente reutilizable por distintas aplicaciones informáticas. Su utilización ahorra tiempo y esfuerzo durante el desarrollo de soluciones informáticas.

Una capa de abstracción de BD tiene las siguientes desventajas:

- Velocidad: Reduce la velocidad en dependencia de la cantidad de código adicional que tiene que ser ejecutado. Mientras más se abstraiga de las interfaces nativas de los SGBD, más lento será el

rendimiento general. Por esta razón se debe analizar su utilización o no en entornos en los cuales se requiera un alto rendimiento.

- Dependencia: Proporciona una nueva dependencia funcional para un sistema informático ya que con el tiempo puede quedar obsoleta o no compatible.
- Operaciones enmascaradas: Puede limitar el número de operaciones disponibles de una BD a un subconjunto de las operaciones admitidas por las BD soportadas.

1.5 Análisis de otras soluciones existentes

Inicialmente, los programadores implementaban el acceso a los datos utilizando las librerías nativas para los SGBD. De esta forma se accedía rápida y directamente a una BD, simplemente escribiendo el código necesario para acceder a la misma. Sin embargo, esto implicaba por parte de los programadores, el aprendizaje de las interfaces de programación para cada sistema gestor existente. También implicaba modificaciones en el código escrito al cambiar la tecnología de BD utilizada.

Como solución a estos problemas surgieron distintas tecnologías de abstracción de BD, las cuales con el paso de los años han evolucionado. Actualmente, los desarrolladores de aplicaciones disponen de numerosas librerías para la abstracción de BD, entre las que se encuentra la interfaz de bajo nivel ODBC, la librería OLEDB, la tecnología ADO.NET, el marco de trabajo libdbi y el módulo QtSql. Cada una tiene sus propias características según el tipo de aplicación a desarrollar y el tipo de BD a acceder.

ODBC (*Microsoft Open Database Connectivity*) es una interfaz de bajo nivel y alto rendimiento programada en C. Fue desarrollada por la corporación Microsoft para posibilitar que las aplicaciones informáticas accedan a los datos de varios SGBD a través de una sola interfaz. Los usuarios pueden incorporar drivers, los cuales actuarán como interfaz entre una aplicación y un SGBD específico (Microsoft 2011). Esta librería ofrece un bajo nivel de abstracción en cuanto a la manipulación de los datos almacenados, por lo que no será utilizada en el desarrollo de la aplicación GeolMin.

OLEDB (*Object Linking and Embedding for Databases,* Enlace e incrustación de objetos para bases de datos) forma parte de las tecnologías MDAC (*Microsoft Data Access Components*). Permite tener acceso a diferentes fuentes de información de manera uniforme. Conceptualmente se divide en consumidores y proveedores. El consumidor es la aplicación que requiere acceso a los datos. El proveedor es el

componente de software que expone una interfaz OLEDB a través del uso del *Component Object Model* (COM) (Microsoft 2011). Esta librería no será utilizada en el desarrollo de la aplicación GeolMin debido a que solo se encuentra disponible para sistemas operativos Windows, y se distribuye bajo los términos de las licencias MS-EULA (*End User License Agreement*).

ADO.NET es una tecnología desarrollada por Microsoft como evolución de la tecnología ADO (*ActiveX Data Objects*). Provee un conjunto de clases que facilitan el acceso a datos provenientes de diferentes fuentes. Se usa en los entornos de programación de la plataforma .NET para manejar BD tanto en aplicaciones de escritorio para Windows como en aplicaciones Web (Joshi, Dickinson et al. 2001). Se encuentra disponible solo para sistemas operativos Windows y se distribuye bajo los términos de la licencia MS-EULA, razones por las cuales no será utilizada para el acceso a los datos del producto GeolMin.

Libdbi es un marco de trabajo que implementa una capa de abstracción de BD en C. Permite establecer múltiples conexiones a BD de forma simultánea sin importar el tipo de servidor al cual se establece la conexión. Provee un driver para cada tipo de servidor de BD. Es un software libre el cual puede ser modificado bajo los términos de la licencia *GNU Lesser General Public (*Parker *and Hoenicka 2005)*. Este marco de trabajo no se utilizará ya que no cuenta con una comunidad activa que le brinde soporte y no se encuentra bien documentado.

QtSql constituye un módulo para BD relacionales ofrecido por el marco de trabajo Qt. Soporta la mayoría de las BD relacionales, entre ellas MySQL, Oracle, PostgreSQL, SyBase, DB2, SQLite e Interbase (Thelin 2007). Ofrece un conjunto de clases que facilitan el acceso y manipulación de los datos con un buen nivel de abstracción. Sin embargo, obliga a emplear el marco de trabajo Qt, lo que traería problemas si en el futuro se decide cambiar Qt por otra librería o se decide dejar de usarlo. Por esta razón no se utilizará este módulo para la abstracción de BD.

1.6 Conclusiones parciales

La implementación del acceso a los datos del sistema GeolMin, de forma que se brinde soporte a distintos SGBD, es una tarea difícil para los programadores. Las librerías existentes para la abstracción de BD no son compatibles o no satisfacen los requisitos para ser integradas a dicho sistema. Por esta razón, es necesaria la creación de un componente de abstracción de BD que garantice la independencia de la aplicación con respecto al SGBD empleado.

Capítulo 2. Tendencias y tecnologías

2.1 Introducción

En este capítulo se analizan las tendencias y tecnologías a utilizar en el desarrollo del componente. Se hace énfasis en la metodología de desarrollo de software y el modelado de software, incluyendo el lenguaje y la herramienta CASE que soporta dicho modelado. También se fundamenta el lenguaje de programación, el entorno de desarrollo y las librerías a emplear, así como los SGBD que soportará el componente.

2.2 Metodologías de desarrollo de software

Una metodología de desarrollo de software constituye un conjunto de procedimientos, actividades y técnicas que guían a los desarrolladores en la construcción de software. Su aplicación tributa a garantizar una alta calidad de los productos. Existen dos grandes grupos de metodologías: las robustas o tradicionales y las ágiles o ligeras.

2.2.1 Metodologías robustas

Las metodologías robustas se centran en la organización y control del proceso de desarrollo de software. Requieren de un equipo de trabajo mayor a los propuestos por las alternativas ágiles y generan mayor número de artefactos. Se caracterizan por el uso exhaustivo de documentación a lo largo de todo el proceso de desarrollo y son muy efectivas en proyectos grandes. Dentro de este tipo de metodologías se encuentra el Proceso Unificado de Rational (RUP).

Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo es un "marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto". Su objetivo consiste en asegurar la producción de software de alta calidad que satisfaga los requerimientos de los clientes dentro del cronograma y presupuesto establecido (Jacobson, Booch et al. 2000).

Se caracteriza principalmente por estar dirigido por casos de uso, centrado en la arquitectura y ser iterativo e incremental. Se basa en componentes de software reutilizables e interconectados a través de interfaces bien definidas. Utiliza el lenguaje UML para representar visualmente los planos del software (Jacobson, Booch et al. 2000).

El proceso de desarrollo se divide en ciclos, los cuales concluyen con una versión del producto. Cada ciclo se divide en cuatro fases: Inicio, Elaboración, Construcción y Transición. Durante estas fases tienen lugar nueve flujos de trabajo: Modelado del negocio, Requerimientos, Análisis y diseño, Implementación, Prueba, Despliegue, Administración de configuración y cambios, Administración del proyecto, y Ambiente (Jacobson, Booch et al. 2000).

La metodología RUP es poco factible para aplicarse en proyectos de corta duración y de pocos integrantes, debido a que propone muchos roles y artefactos. Además, se genera demasiada documentación, resultando inadecuada para entornos en los cuales se producen constantes cambios en los requisitos. Por estas razones se considera que esta metodología no es apropiada para el desarrollo del componente.

2.2.2 Metodologías ágiles

Las metodologías ágiles tienen como objetivo la entrega rápida de software que satisfaga las necesidades del cliente. Se orientan a equipos de tamaño pequeño y conciben al cliente como parte del equipo de trabajo. Se adaptan rápidamente a los cambios que se producen a lo largo del proyecto. Además, se caracterizan por definir pocos roles y artefactos.

Extreme Programming (XP)

La metodología XP se centra en las relaciones interpersonales como factor clave para lograr el éxito en el desarrollo del software. Permite mantener un nivel mínimo de documentación con vistas a agilizar el desarrollo del software. Es aplicable a proyectos con requisitos imprecisos y cambiantes. Propone un ciclo de vida dividido en seis fases: Exploración, Planificación de la entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto (Amaro Calderón and Valverde Rebaza 2007).

Promueve la comunicación entre los desarrolladores y clientes, permitiendo una rápida retroalimentación de los conocimientos. Recomienda que la programación se realice en parejas, contribuyendo a la disminución de la tasa de errores y al aseguramiento de la calidad. Exige la presencia constante del cliente en el lugar de desarrollo, siendo este el responsable de orientar el trabajo de forma que le aporte mayor valor al negocio (Amaro Calderón and Valverde Rebaza 2007).

Durante el desarrollo del componente no se puede garantizar la presencia constante del cliente por lo que se decide no usar esta metodología. Además, se requiere una adecuada documentación, tributando a la reutilización del componente en aplicaciones futuras, así como el desarrollo de nuevas versiones del mismo.

Feature Driven Development (FDD)

El Desarrollo Conducido por Características (FDD, *Feature Driven Development*) es una metodología ágil orientada a las funcionalidades del software. Tiene como objetivo satisfacer a los desarrolladores y clientes sin afectar el proceso de desarrollo de software (Calabria 2003). Fue diseñada por Peter Coad, Eric Lefebvre y Jeff DeLuca. Estos autores definieron una característica o rasgo como una función valuada por el cliente, la cual puede implementarse en aproximadamente dos semanas (Pressman 2005).

Es aplicable a proyectos de corta duración, es decir, menos de un año. Se basa en un proceso de iteraciones cortas, contribuyendo a la obtención de resultados tangibles de forma periódica. A diferencia de otras metodologías, se centra en el diseño y construcción de la aplicación y no en la obtención de requerimientos. Tiene en cuenta la calidad del software incluyendo un monitoreo constante del proyecto (Calabria 2003).

Propone un ciclo de vida compuesto por cinco procesos secuenciales (Figura 3). Los tres primeros se realizan al inicio del proyecto y los dos últimos de forma iterativa (Calabria 2003).

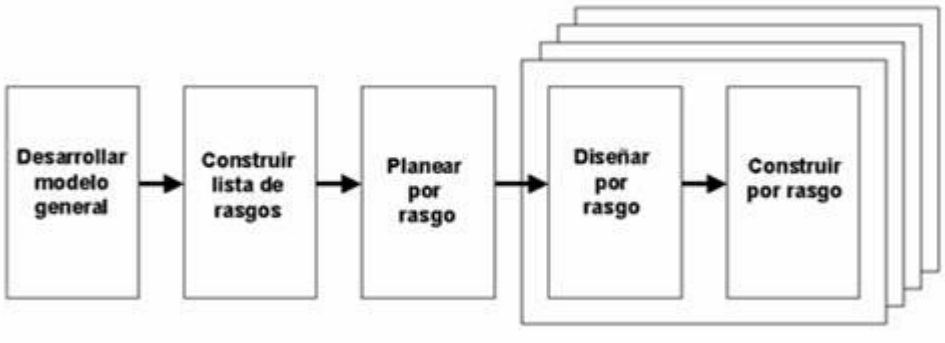

Figura 3. Procesos de la metodología FDD.

- **Desarrollo de un modelo global**

Tiene como objetivo lograr un conocimiento global de la aplicación a construir, así como un entendimiento del negocio. Para esto los expertos del dominio presentan un ensayo donde describen a los miembros del equipo el sistema a construir. Luego se divide el dominio global en diferentes áreas, las cuales son asignadas a subgrupos. Se realiza un ensayo del dominio para cada área y a partir de estos modelos el equipo construye un modelo global.

- **Construcción de la lista de rasgos**

Se identifican las características o rasgos, y posteriormente se agrupan y priorizan. La lista de rasgos se divide en áreas temáticas, actividades del negocio y rasgos necesarios para dar cumplimiento a cada actividad.

- **Planificación por rasgo**

A partir de las características identificadas se diseña un cronograma para el posterior diseño y construcción de la aplicación, teniendo en cuenta la prioridad y dependencia de las funcionalidades. También se seleccionan las características a incluir en cada iteración.

- **Diseño y construcción por rasgo**

Para cada iteración se diseñan y construyen las clases involucradas en los rasgos correspondientes. También se realizan tareas de inspección al diseño y al código, así como pruebas unitarias y de integración.

La metodología FDD es aplicable a proyectos cortos y de pocos integrantes. No exige la presencia constante del cliente. Genera una documentación aceptable y menor en comparación con RUP. La división del proyecto en pequeñas unidades facilita el seguimiento del avance del mismo. Por estas razones se decidió usar dicha metodología para el desarrollo del componente.

2.3 Modelado de software

El modelado de software permite capturar las partes esenciales del sistema y manejar la complejidad de las aplicaciones a construir. Se realiza a través de la utilización de una herramienta CASE (*Computer Aided Software Engineering*) empleando un lenguaje de modelado.

2.3.1 Lenguaje Unificado de Modelado (UML)

UML (*Unified Modeling Language*) es un "lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software". Constituye una notación estándar para la construcción de modelos orientados a objetos, respaldado por el OMG (*Object Management Group*). Es adoptado por numerosas herramientas CASE, entre las que se encuentra Visual Paradigm. Unifica las técnicas de modelado existentes en el momento de su creación: la de Grady Booch, el OMT (*Object Modeling Technique*) y el OOSE (*Object-Oriented Software Engineering*) (Larman 2002).

Está compuesto por un conjunto de elementos gráficos los cuales se combinan para formar modelos. Estandariza 9 tipos de diagramas para la representación gráfica de diferentes aspectos del sistema. A través de estos diagramas se describen los elementos que deben existir y qué debe hacer el sistema, antes de su construcción. También sirven para documentar el software desarrollado (Hernández Orallo 2002).

Es un lenguaje para la construcción de modelos; no guía al desarrollador en la forma de crear los modelos, ni especifica el proceso de desarrollo a adoptar (Larman 2002). Es independiente del lenguaje de programación, de ahí que los diseños realizados pueden ser implementados en cualquier lenguaje. Permite comunicar la estructura del sistema y especificar su comportamiento, logrando una mejor comprensión del mismo (Hernández Orallo 2002).

2.3.2 Herramientas CASE

Las herramientas CASE son aplicaciones informáticas que proporcionan ayuda automatizada en las actividades del proceso de desarrollo de software. Tienen como objetivos mejorar la productividad en el desarrollo y mantenimiento de software, así como aumentar la calidad del mismo. Su utilización permite reducir el tiempo y el costo para la construcción de un sistema.

Entre las herramientas CASE más utilizadas se encuentra Rational Rose Enterprise Edition y Visual Paradigm for UML (VP). La primera fue creada por la corporación Rational Software (que actualmente forma parte de IBM), y la segunda fue desarrollada por Visual Paradigm International Ltd. Ambas herramientas soportan el ciclo completo de desarrollo de software, ayudando en la construcción de aplicaciones de calidad en menor tiempo y costo.

Tanto VP como Rational Rose Enterprise Edition permiten el modelado de software empleando UML. VP permite modelar los diagramas usando la versión 2.1 de UML (Visual_Paradigm_International 2011), mientras Rational Rose Enterprise Edition solo permite el diseño de diagramas usando UML 1.x (IBM 2011). Además, ofrecen capacidades de ingeniería directa e inversa, así como la generación de informes. VP también permite la exportación de los diagramas como imágenes.

Rational Rose Enterprise Edition se encuentra publicada bajo licencia comercial, es altamente costosa y solo se ejecuta en plataformas Windows (IBM 2011). En cambio, VP se distribuye bajo las licencias comercial y gratuita, y se puede ejecutar en múltiples plataformas, como por ejemplo Windows, Linux y Mac OS (Visual_Paradigm_International 2011). Teniendo en cuenta las características expuestas anteriormente, se optó por usar Visual Paradigm for UML 6.4 Enterprise Edition para modelar el componente a desarrollar.

2.4 Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) agrupa un conjunto de técnicas que facilitan la producción y mantenimiento de software, permitiendo abordar programas de gran complejidad. Se define como un paradigma de programación que expresa un programa como un conjunto de objetos que colaboran entre ellos para llevar a cabo las acciones (Moreno 2000).

Los objetos son entidades que se controlan a sí mismas, liberando al programador de esta tarea. Ocultan los detalles internos de cómo llevan a cabo las acciones. La POO consiste en definir clases de objetos que cumplen determinadas propiedades (estado o atributos) y comportamientos (métodos). Los atributos y los métodos se encuentran estrechamente relacionados, permitiendo estos últimos modificar los atributos y prestar servicios a otros objetos (Moreno 2000).

La POO intenta resolver algunos problemas que se presentan en la Ingeniería de Software, entre los que se encuentra la portabilidad, la reusabilidad y la extensibilidad. Para ello se basa en características claves como la herencia, el polimorfismo, el encapsulamiento, las interfaces y la ocultación de información. Estos elementos la convierten en una de las mejores opciones para producir software (Rodríguez 2008).

Los lenguajes orientados a objetos son aquellos que implementan los conceptos de la POO. Se pueden clasificar en puros e híbridos. Un lenguaje se considera puro cuando se ajusta completamente a los principios propuestos por la POO y permite trabajar exclusivamente con clases. Un lenguaje se considera híbrido de POO y otro paradigma cuando incorpora en mayor o menor medida facilidades para trabajar con clases (Moreno 2000). En la actualidad numerosos lenguajes de programación soportan la orientación a objetos, entre los cuales se encuentra C++, C# y Java.

2.4.1 Lenguaje de programación C++

"C++ es un lenguaje de programación basado en C que soporta directamente conceptos de la Orientación a Objetos. Retiene los recursos de bajo nivel y la eficiencia de C". Además, elimina algunas dificultades presentes en el C original. Es considerado un lenguaje híbrido ya que en él coexisten la programación estructurada y la POO. Fue creado en 1980 por Bjarne Stroustrup en los Laboratorios Bell; inicialmente se llamaba C++ con clases (Katrib Mora 1997).

Al igual que C, sigue muy ligado al hardware subyacente lo que permite la programación a bajo nivel, aunque también permite lograr un alto nivel de abstracción. Los códigos escritos en C++ pueden ser compilados en la mayoría de los ordenadores y sistemas operativos existentes. Además, ocupan menos memoria en comparación con otros lenguajes y son más rápidos (Zator_Systems 2011).

Incorpora nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, referencias, entre otros operadores y funciones. También ofrece numerosos recursos de la POO como por ejemplo la herencia, el polimorfismo, el encapsulamiento y la sobrecarga de operadores y funciones. Estos elementos permiten una mayor reutilización del código y una alta calidad del software (Katrib Mora 1997).

La utilización de C++ como lenguaje para implementar el componente constituye una restricción. Esto se debe a que el componente se va a integrar al sistema GeolMin, el cual está siendo desarrollado en dicho lenguaje de programación.

2.5 Entornos de Desarrollo Integrado

Un entorno de desarrollo integrado (IDE, *Integrated Development Environment*) constituye un programa informático compuesto por un conjunto de herramientas que emplean los programadores para crear aplicaciones. Generalmente se compone de un editor de código, un compilador, un depurador y un constructor de interfaces de usuario. Puede estar dedicado a un lenguaje de programación específico o proveer un marco de trabajo para múltiples lenguajes de programación.

Para el desarrollo del componente se identificaron como posibles IDE: Visual C++ 2010, C++ Builder 2010 y Qt Creator 2.01. Visual C++ 2010 es un IDE desarrollado por Microsoft para los lenguajes de programación C, C++ y C++/CLI. Constituye uno de los entornos de desarrollo más completos para crear aplicaciones para Windows. Incluye bibliotecas estándares como por ejemplo *Active Template Library* (ATL) y *Microsoft Foundation Class* (MFC). Se distribuye bajo licencia propietaria y es altamente costoso. Además, tiene un alto requerimiento de memoria (Microsoft 2011).

C++ Builder 2010 fue creado por la empresa Embarcadero Technologies. Soporta el Desarrollo Rápido de Aplicaciones (RAD, *Rapid Application Development*) en C/C++, así como las reconocidas bibliotecas

Boost. Permite crear aplicaciones con interfaces complejas. Su compilador es más lento que el de Visual C++ 2010. Se ejecuta en plataformas Windows y se distribuye bajo licencia privativa, aunque su precio es menor al de Visual C++ 2010.

Qt Creator 2.01 es un IDE multiplataforma para el desarrollo de aplicaciones con interfaces gráficas. Fue creado por Trolltech y actualmente es propiedad de Nokia. A diferencia de Visual C++ 2010 y C++ Builder 2010, se puede ejecutar en plataformas Linux, Mac OS X y Windows. Se encuentra publicado bajo tres tipos de licencias: *Qt Commercial Developer License, Qt GNU LGPL v. 2.1* y *Qt GNU GPL v. 3.0*. Su código fuente se encuentra disponible y posee un avanzado editor de código en C++. Se puede integrar a los principales sistemas de control de versiones incluyendo Subversion (Nokia 2011).

Se optó por emplear Qt Creator 2.01 como IDE para la creación del componente teniendo en cuenta las facilidades que ofrece para la edición de código en C++. Además, se ejecuta en múltiples plataformas y posee una amplia documentación.

2.6 Sistemas Gestores de Bases de Datos (SGBD)

2.6.1 SQLite

SQLite es un SGBDR (Sistema Gestor de Bases de Datos Relacionales) contenido en una librería escrita en lenguaje C. Fue creado para proporcionar una manera conveniente para la gestión de los datos sin la sobrecarga que a menudo caracteriza a los SGBD. A diferencia de los SGBD cliente-servidor se incrusta como parte de la aplicación que lo aloja, es decir, el cliente y el servidor se ejecutan en un mismo proceso. Esta característica constituye una de sus principales ventajas ya que no requiere servidor ni configuración (Owens 2006).

Almacena las BD con todos sus objetos en un solo archivo ordinario. A pesar del poco espacio que ocupa en memoria, ofrece numerosas características y funcionalidades. Implementa un amplio subconjunto del estándar ANSI SQL92, incluyendo las transacciones, vistas, restricciones de comprobación, subconsultas, disparadores, índices, entre otras. También tiene características únicas, tales como BD en memoria, tipado dinámico y resolución de conflictos (Owens 2006).

Su utilización facilita la implementación de las aplicaciones al simplificar la administración de las BD, razón por la cual tiene amplia aceptación por las empresas para realizar demostraciones o pruebas. Su código se encuentra en dominio público. También se caracteriza por ser muy portátil, rápida, confiable y fácil de usar (Owens 2006). El componente a desarrollar empleará la librería **sqlite3** con vistas a brindar soporte a dicho gestor.

2.6.2 PostgreSQL

PostgreSQL es un SGBD objeto-relacional desarrollado por la comunidad internacional PGDG (*PostgreSQL Global Development Group*). Se distribuye bajo los términos de la licencia BSD y es considerado uno de los SGBD de código abierto más potentes. Su código está disponible sin costo alguno tributando a su extensibilidad (Douglas and Douglas 2003). Se encuentra disponible para múltiples plataformas como Windows, Linux, Mac OS X, Solaris, FreeBSD, entre muchas otras. Además, se caracteriza por utilizar un modelo cliente-servidor (Matthew and Stones 2005).

Soporta los estándares SQL92, SQL99, SQL2003 y SQL2008. Funciona bien con grandes cantidades de datos y permite una alta concurrencia de usuarios accediendo al sistema de forma simultánea. Soporta el almacenamiento de objetos multimedia tales como gráficos, videos y sonidos. Cuenta con APIs para programar en distintos lenguajes como por ejemplo C/C++, Java, .Net, Perl, Python, Ruby, PHP, Qt, entre otros (Martínez 2010).

PostgreSQL incluye transacciones, subconsultas, vistas, integridad referencial, tipos definidos por el usuario, herencia, control de concurrencia, procedimientos almacenados, disparadores, entre otras características. Es totalmente ACID (***A**tomicity, **C**onsistency, **I**solation and **D**urability*), es decir, cuenta con las funcionalidades necesarias para que sus transacciones cumplan las propiedades de atomicidad, consistencia, aislamiento y durabilidad (PGDG 2011). Para brindar soporte a dicho SGBD se empleará la librería **libpqxx**, considerada la interfaz oficial de C++ para PostgreSQL.

2.7 Lenguaje Estructurado de Consultas (SQL)

SQL (*Structured Query Language*) es un lenguaje estructurado de consultas que permite acceder, consultar, describir y manipular los datos en los SGBDR. Constituye una evolución del lenguaje SEQUEL desarrollado en 1976 por Donald D. Chamberlin. Fue creado en un centro de investigación de IBM en los años setenta (Camallea 2004). El lenguaje SQL tiene tres componentes fundamentales: el lenguaje de definición de datos (DDL), el lenguaje de manipulación de datos (DML) y el lenguaje de control de datos (DCL). Está compuesto por un conjunto de comandos, cláusulas, operadores y funciones de agregado.

Constituye un estándar ANSI (*American National Standards Institute*) que consta de varias versiones y sufre revisiones periódicamente. No está soportado por ningún producto comercial, cada SGBD determina las partes del lenguaje que implementará y cómo lo hará, incluso pueden incorporar funcionalidades que no pertenecen al estándar (Ezequiel Rozic 2004). Actualmente es considerado el lenguaje de los negocios porque es el lenguaje relacional escogido por la mayoría de los sistemas comerciales.

2.8 Conclusiones parciales

La metodología FDD guiará el proceso de desarrollo del componente garantizando una alta calidad del mismo. La utilización del Visual Paradigm for UML 6.4 Enterprise Edition para el modelado del software usando UML, permitirá captar las partes esenciales del componente a construir y posibilitará reducir el tiempo de desarrollo a través de la automatización de algunas tareas.

El lenguaje C++ se caracteriza por tener un alto grado de optimización en velocidad y uso de la memoria. El IDE Qt Creator 2.01 ofrece un avanzado editor de código que facilitará la implementación del componente usando C++ como lenguaje de programación. El componente brindará soporte a los gestores SQLite y PostgreSQL a través de las librerías sqlite3 y libpqxx, respectivamente.

Capítulo 3. Características del componente

3.1 Introducción

En este capítulo se desarrolla un modelo global con vistas a lograr un entendimiento del negocio y del componente a construir. También se identifican las características o funcionalidades que el componente debe tener, plasmándolas en un listado. Finalmente, se diseña un cronograma para el posterior diseño y construcción del componente.

3.2 Desarrollo del modelo global

3.2.1 Ensayo del dominio

El sistema GeolMin necesita un mecanismo que le permita manipular la información con independencia del tipo de BD utilizada. Para lograrlo se requiere desarrollar un componente de abstracción de BD que permita la conexión a varios SGBD (SQLite y PostgreSQL), así como la ejecución de consultas SQL y la realización de transacciones.

Para establecer la conexión a una BD, primeramente se debe especificar el proveedor de BD a utilizar y luego configurar los datos de conexión. En el caso de una BD SQLite solo se necesita conocer la dirección en la que se encuentra la misma. Sin embargo, para establecer la conexión a una BD PostgreSQL se requiere conocer el servidor donde se encuentra alojada, el puerto, el nombre de la BD, así como el usuario y la contraseña. Una vez finalizadas las operaciones sobre la BD se debe desconectar.

Las consultas constituyen un conjunto de instrucciones que permiten la definición, manipulación y recuperación de los datos. Estas se clasifican en consultas de selección o de acción. Las primeras son aquellas que devuelven un conjunto de registros como resultado de seleccionar datos de una o varias tablas. Las segundas realizan acciones específicas sobre los datos almacenados en una tabla, las cuales no devuelven ningún registro o valor. Entre las consultas de acción se encuentra la creación, modificación y eliminación de tablas, la inserción, actualización y eliminación de datos.

Una transacción agrupa un conjunto de acciones que son tratadas como una actividad indivisible y cuya ejecución conserva la consistencia de la BD. Estas acciones se realizan en su totalidad o ninguna es ejecutada, aunque el sistema falle a mitad del proceso. Si la transacción se realiza satisfactoriamente, debe ser confirmada para aplicar los cambios provocados en la BD. En caso de producirse errores, los cambios ocasionados no se registran. Con este fin se proporcionan las operaciones BEGIN TRANSACTION (iniciar transacción), COMMIT (confirmar) y ROLLBACK (deshacer).

El componente también deberá proveer una sintaxis única para la ejecución de consultas simples, con independencia de la BD empleada. Para lograr esto el componente debe permitir la construcción de consultas SQL simples a través de la invocación a distintos métodos.

3.2.2 Modelo global

Un modelo global consiste en la construcción de un diagrama de clases que representa los objetos más importantes en el dominio del problema y las relaciones entre estos. En este diagrama se pueden incluir relaciones de herencia y generalización/ especialización. También se incluyen operaciones que describen en forma de rasgos el comportamiento de los objetos, sin especificar conveniencias de programación. Este modelo ayuda a comprender los conceptos que utilizan los usuarios y con los cuales deberá trabajar la aplicación.

A continuación se muestra el modelo global del componente de abstracción de BD. En el mismo se representa un proveedor de BD que puede ser SQLite o PostgreSQL. Cada proveedor cuenta con una interfaz de BD que permite establecer la conexión a la BD correspondiente, ejecutar las consultas SQL y realizar las transacciones. Las consultas pueden ser de acción o de selección. Las últimas devuelven un resultado conformado por un conjunto de registros. Una transacción consiste en la ejecución de un conjunto de consultas tratadas como una actividad indivisible. En el modelo también se representa una clase llamada ObjetoConsulta, la cual permite la construcción de consultas SQL simples a través de la invocación a distintos métodos.

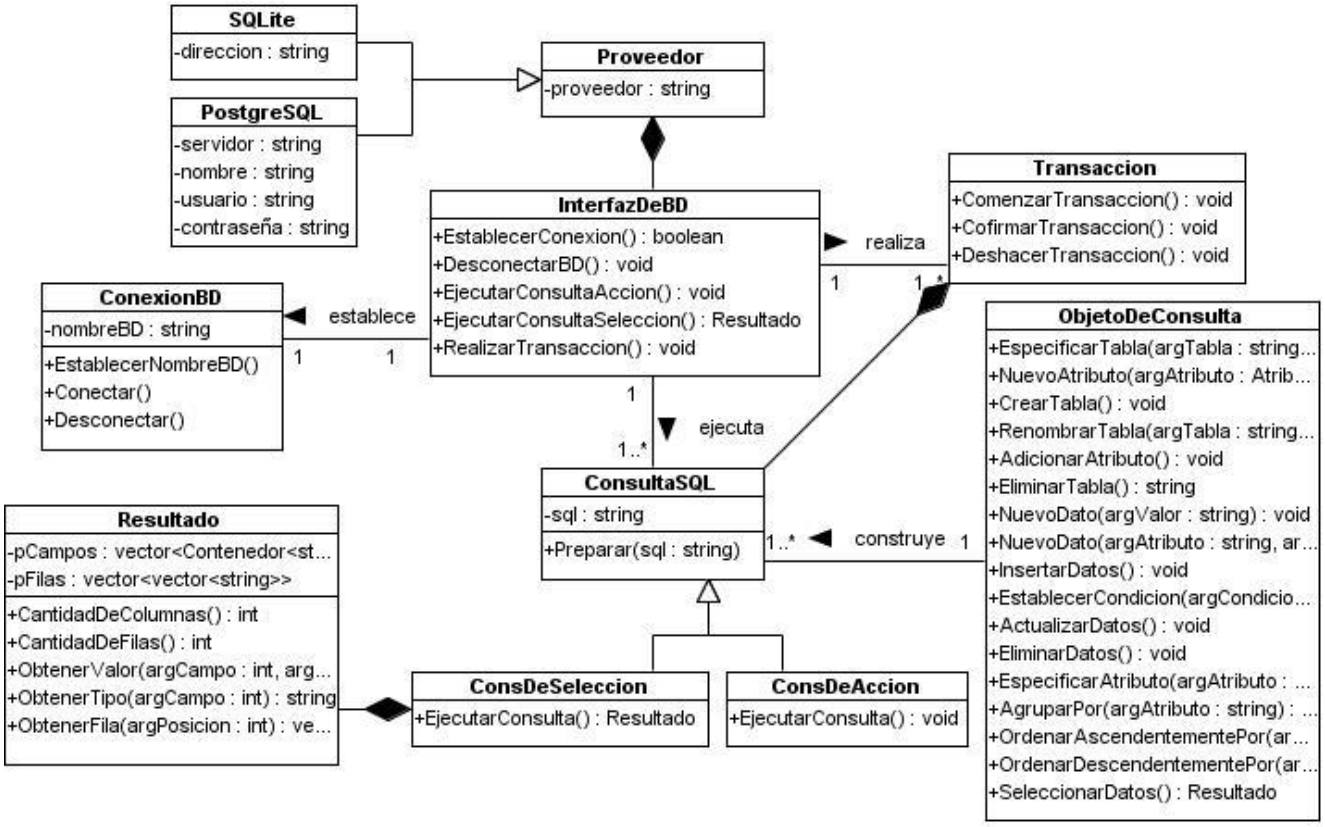

Figura 4. Modelo global del componente de abstracción de BD.

3.3 Construcción de la lista de rasgos

3.3.1 Lista de rasgos

A partir del modelo global se identificaron un conjunto de rasgos que resumen las funcionalidades del sistema. Posteriormente se realizó la priorización de estos en base a la satisfacción del cliente, teniendo en cuenta las prioridades establecidas por FDD para los rasgos: A (debe estar), B (deseable que esté), C (deseable si se puede) y D (deseable en el futuro). Luego se organizaron jerárquicamente en el listado que se muestra a continuación.

Tabla 1. Lista de rasgos del componente de abstracción de BD.

Lista de rasgos		
Área temática	**Actividad del negocio**	**Rasgos**
Abstracción de BD	Conectar/Desconectar una BD	Establecer el proveedor de BD: SQLite o PostgreSQL. ***(A)***
		Configurar los datos de conexión a una BD. ***(A)***
		Conectar a una BD. ***(A)***
		Desconectar una BD. ***(A)***
	Ejecutar consultas SQL	Ejecutar consultas de acción: ***(A)***
		- Insertar datos en una tabla.
		- Actualizar datos de una tabla.
		- Eliminar datos de una tabla.
		- Crear una tabla en la BD.
		- Modificar una tabla de la BD.
		- Eliminar una tabla de la BD.
		Ejecutar consultas de selección. ***(A)***
	Realizar transacciones	Comenzar una transacción. ***(A)***
		Confirmar una transacción. ***(A)***
		Deshacer una transacción. ***(A)***
	Construir consultas SQL simples	Construir consultas de inserción de datos. ***(B)***
		Construir consultas de actualización de datos. ***(B)***
		Construir consultas de selección de datos. ***(B)***
		Construir consultas de eliminación de datos. ***(B)***
		Construir consultas de creación de una tabla. ***(B)***
		Construir consultas de modificación de una tabla. ***(B)***
		Construir consultas de eliminación de una tabla. ***(B)***

3.3.2 Rasgos no funcionales

Seguidamente se presentan los rasgos no funcionales del componente, es decir, las propiedades o cualidades que debe tener.

Usabilidad

1. El componente deberá ser fácil de usar por los programadores del sistema GeolMin, aunque estos tengan un bajo dominio del lenguaje SQL.

Seguridad

2. El componente realizará una adecuada gestión de los errores que puedan ocurrir, garantizando la estabilidad e integridad del mismo.

Restricciones de diseño

3. El sistema será desarrollado usando C++ como lenguaje de programación.

Ayuda y soporte técnico

4. El componente debe contar con un manual de soporte adjunto al mismo, en el cual se expongan sus principales funcionalidades.

3.4 Planeación por rasgo

El componente será desarrollado en 3 iteraciones, planificadas de la siguiente manera:

- *Iteración 1*: Rasgos involucrados en la actividad "Conectar/Desconectar una BD".
- *Iteración 2*: Rasgos involucrados en las actividades "Ejecutar consultas SQL" y "Realizar transacciones".
- *Iteración 3*: Rasgos involucrados en la actividad "Construir consultas SQL simples".

A continuación se presenta el cronograma de diseño e implementación de los rasgos identificados. Para realizar dicha planificación se tuvo en cuenta la prioridad y la dependencia existente entre los rasgos.

Área temática: Abstracción de BD.

Tabla 2. Planeación de los rasgos: "Conectar/Desconectar una BD".

Conectar/Desconectar una BD			**Rasgos:** 4
Descripción	**Diseño**	**Construcción**	**Pruebas**
	Plan	**Plan**	**Plan**
Establecer el proveedor de BD (SQLite/ PostgreSQL)	01/03/11	02/03/11	03/03/11
Configurar los datos de conexión a una BD	04/03/11	05/03/11	07/05/11
Conectar a una BD	08/03/11	9/03/11	10/03/11
Desconectar una BD	11/03/11	12/03/11	14/03/11

Tabla 3. Planeación de los rasgos: "Ejecutar consultas SQL".

Ejecutar consultas SQL			**Rasgos:** 2
Descripción	**Diseño**	**Construcción**	**Pruebas**
	Plan	**Plan**	**Plan**
Ejecutar consultas de acción	15/03/11	16/03/11	17/03/11
Ejecutar consultas de selección	18/03/11	19/03/11	21/03/11

Tabla 4. Planeación de los rasgos: "Realizar transacciones".

Realizar transacciones			**Rasgos:** 3
Descripción	**Diseño**	**Construcción**	**Pruebas**
	Plan	**Plan**	**Plan**
Comenzar una transacción	22/03/11	23/03/11	24/03/11
Finalizar una transacción	25/03/11	26/03/11	28/03/11
Deshacer una transacción	29/03/11	30/03/11	31/03/11

Tabla 5. Planeación de los rasgos: "Construir consultas SQL simples".

Construir consultas SQL simples		**Rasgos:** 7	
Descripción	**Diseño**	**Construcción**	**Pruebas**
	Plan	**Plan**	**Plan**
Construir consultas de inserción de datos	01/04/11	02/04/11	04/04/11
Construir consultas de actualización de datos	01/04/11	02/04/11	04/04/11
Construir consultas de selección de datos	05/04/11	06/04/11	07/04/11
Construir consultas de eliminación de datos	08/04/11	09/04/11	11/04/11
Construir consultas de creación de una tabla	08/04/11	09/04/11	11/04/11
Construir consultas de modificación de una tabla	12/04/11	13/04/11	14/04/11
Construir consultas de eliminación de una tabla	12/04/11	13/04/11	14/04/11

3.5 Conclusiones parciales

En este capítulo se llevaron a cabo los tres primeros procesos que propone la metodología FDD para el desarrollo de software. Como resultado se obtuvo un modelo global del sistema que permitió la comprensión del componente a construir y del negocio en el cual está embebido. Se elaboró una lista de rasgos que representan las funcionalidades del componente, de acuerdo a las necesidades del cliente. También se identificaron los rasgos no funcionales que debe cumplir, así como sus principales clases. Se planificaron los procesos de diseño y construcción de cada uno de los rasgos identificados. Se definieron 3 iteraciones para la creación del componente, propiciando la obtención de resultados tangibles a corto plazo.

Capítulo 4. Diseño del componente

4.1 Introducción

En este capítulo se define la arquitectura de software y los patrones a utilizar en el diseño e implementación de la solución. Seguidamente se modela el componente a través de un diagrama de clases del diseño, describiéndose las clases involucradas y proporcionando los detalles necesarios para la posterior construcción de la solución propuesta.

4.2 Arquitectura basada en componentes

La arquitectura de software es un elemento fundamental en la construcción de cualquier sistema informático. Esta se define como un "marco conceptual completo que describe su forma y estructura" (Jerrold Grochow), es decir sus componentes y la manera en que lo integran. La arquitectura de software permite la comunicación entre las partes involucradas en el desarrollo de un sistema. Además, incluye decisiones fundamentales relativas al diseño de aplicaciones, las cuales impactan el trabajo de ingeniería de software e influyen en el éxito final (Pressman 2005).

La arquitectura basada en componentes se enfoca en la descomposición de un sistema en componentes funcionales, los cuales exponen interfaces que permiten la comunicación entre dichos componentes. Las interfaces juegan un papel fundamental ya que permiten acceder a las funcionalidades sin revelar los detalles internos de la implementación. De esta forma se proporciona un alto nivel de abstracción en comparación con la orientación a objetos (Peláez 2009).

Un componente de software se define como un objeto de software diseñado para cumplir con determinado propósito. Los componentes se caracterizan por ser reutilizables, de ahí que puedan ser utilizados por diferentes aplicaciones. También se caracterizan por ser fácilmente extensibles. En el diseño de los mismos se tienen en cuenta 5 principios fundamentales: reusabilidad, sin contexto específico, extensibilidad, encapsulamiento e independencia (Peláez 2009).

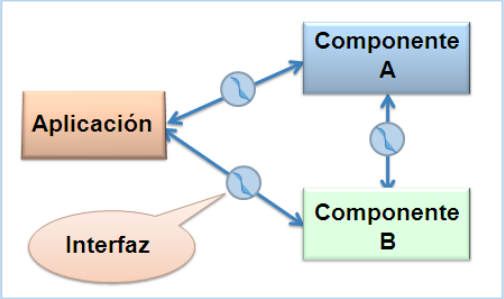

Figura 5. Representación de la arquitectura basada en componentes.

La arquitectura basada en componentes constituye uno de los mecanismos de software más efectivos para la creación de grandes sistemas. La misma ofrece numerosos beneficios, entre los que se encuentra la facilidad de instalación, permitiendo que los componentes puedan ser reemplazados por otras versiones sin impactar al resto del sistema. La utilización de componentes de software también disminuye el tiempo de desarrollo y permite gestionar la creciente complejidad de las soluciones informáticas (Peláez 2009).

El componente desarrollado emplea la arquitectura basada en componentes. El mismo utiliza otros componentes prefabricados que le permiten acceder y manipular la información contenida en una BD específica. Además, permite acceder a sus funcionalidades de una forma sencilla y transparente, a través de una interfaz bien definida. Dicha interfaz expone los métodos necesarios para establecer la conexión a una BD, así como ejecutar consultas y transacciones.

4.3 Patrones de diseño

Los patrones ofrecen soluciones a problemas específicos y comunes, que se presentan en determinado contexto durante la creación de software. No expresan nuevos principios, sino que proponen soluciones demostradas y basadas en la experiencia acumulada por los ingenieros de software. La aplicación de patrones contribuye a mejorar la calidad del diseño y la implementación de sistemas. También tributa a la reutilización de las clases y del propio diseño, evitando que se reinvente la rueda en cada proyecto nuevo.

Los patrones de diseño ofrecen soluciones a problemas que se presentan en el diseño de sistemas orientados a objetos. Permiten establecer un lenguaje común entre diseñadores y desarrolladores, cambiando el nivel de abstracción a colaboraciones entre clases y permitiendo comunicar experiencia

sobre dichos problemas y soluciones. Entre los patrones de diseño más utilizados se encuentran los patrones GRASP y los patrones GoF (Pérez Valls 2009).

La asignación de responsabilidades es un aspecto fundamental en el diseño de sistemas orientados a objetos. Existe un conjunto de principios que ofrecen soluciones a los problemas relacionados con la asignación de responsabilidades, llamados patrones GRASP (*General Responsibility Assignment Software Patterns*). En el desarrollo del componente se aplicaron los patrones: Experto, Creador, Bajo Acoplamiento, Alta Cohesión, Controlador, Polimorfismo, Fabricación Pura e Indirección.

El patrón Experto plantea que las responsabilidades deben asignarse a las clases que contienen la información necesaria para dar cumplimiento a las mismas. El patrón Creador ayuda a decidir qué clase es responsable de la creación de los objetos. El patrón Bajo Acoplamiento recomienda la asignación de responsabilidades de modo que no se incremente la dependencia entre las clases y se facilite su reutilización. El patrón Alta Cohesión plantea que las responsabilidades de una clase deben estar estrechamente relacionadas. El patrón Controlador ayuda a decidir qué clase debería controlar un evento del sistema (Larman 2002).

El patrón Polimorfismo recomienda asignar las responsabilidades de un comportamiento, utilizando operaciones polimórficas cuando los comportamientos varíen según el tipo de clase. El patrón Fabricación Pura propone asignar un conjunto de responsabilidades relacionadas a una clase artificial, es decir, una clase que no existe en el dominio del problema. El patrón Indirección plantea asignar la responsabilidad a un objeto que actúe como intermediario entre otros componentes o servicios, evitando que se acoplen directamente (Larman 2002).

Los patrones de diseño GoF fueron creados por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, conocidos como "La Pandilla de los Cuatro" (*Gang of Four*). Se clasifican en patrones creacionales, estructurales y de comportamiento. Los creacionales resuelven problemas relativos a la creación de objetos, mientras los estructurales ofrecen solución a problemas relativos a la composición de objetos. Por su parte, los patrones de comportamiento resuelven los problemas referentes a la interacción entre los objetos (Cooper 1998).

Durante el diseño del componente se aplicaron los patrones creacionales Solitario (*Singleton*) y Fábrica Abstracta y el patrón estructural Fachada. El patrón Fachada proporciona una interfaz unificada que representa un subsistema y permite acceder a las funcionalidades de un conjunto de clases (Larman 2002). Este patrón se utilizó en la clase *BaseDeDatos*, la cual se comporta como una fachada del componente CABD proporcionando acceso a las funcionalidades de las clases *Conexion*, *Consulta* y *Transaccion*. Esta clase actúa como intermediaria entre los clientes y el componente, ocultando la complejidad del mismo.

El patrón Solitario garantiza la existencia de una sola instancia de una clase proporcionando un punto de acceso global a la misma (Larman 2002). Este patrón es utilizado para proveer una instancia única de la clase *BaseDeDatos*, la cual permite el acceso y manipulación de los datos almacenados (Figura 6). La propia clase garantiza la creación de una sola instancia y proporciona un punto global de acceso a la misma a través del método *Instancia*. Para lograr esto el constructor de la clase se declaró privado de modo que no pueda ser instanciada directamente. De esta forma se restringe el número de conexiones establecidas a una misma BD.

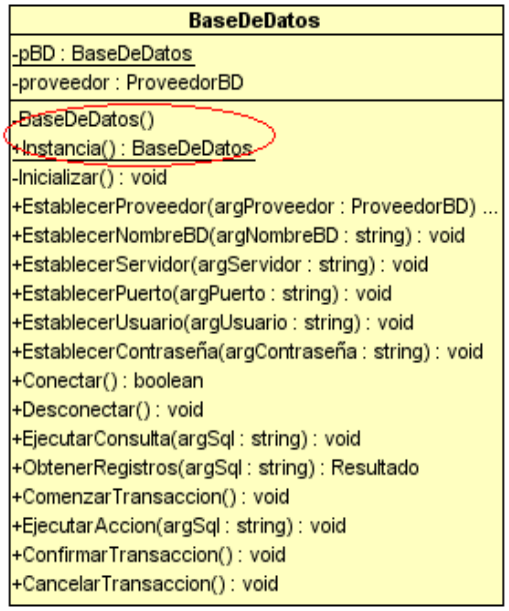

Figura 6. Aplicación de los patrones Fachada y Solitario a la clase BaseDeDatos.

El patrón Fábrica Abstracta proporciona una interfaz para la creación de familias de objetos interrelacionados, sin especificar las instancias concretas (Larman 2002). Se aplicó para garantizar la creación de los objetos necesarios para interactuar con un sistema gestor específico (Figura 7). Para lograr esto cada sistema gestor soportado por el componente cuenta con una clase que hereda de la clase *FabricaAbstracta*, la cual se encarga de crear los objetos: *Conexion*, *Consulta* y *Transaccion*, necesarios para interactuar con el mismo.

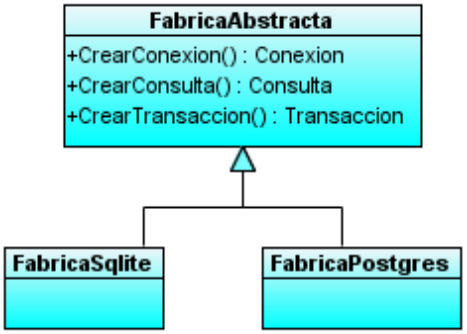

Figura 7. Aplicación del patrón Fábrica Abstracta.

4.4 Diagrama de clases del diseño

Los diagramas de clases del diseño describen gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Las clases del diseño son especificadas empleando la sintaxis del lenguaje de programación seleccionado. Las relaciones entre las clases involucradas tienen implicaciones directas en la implementación, y los métodos se corresponden directamente con los métodos a implementar. En la figura 8 se muestra el diagrama de clases del diseño del componente.

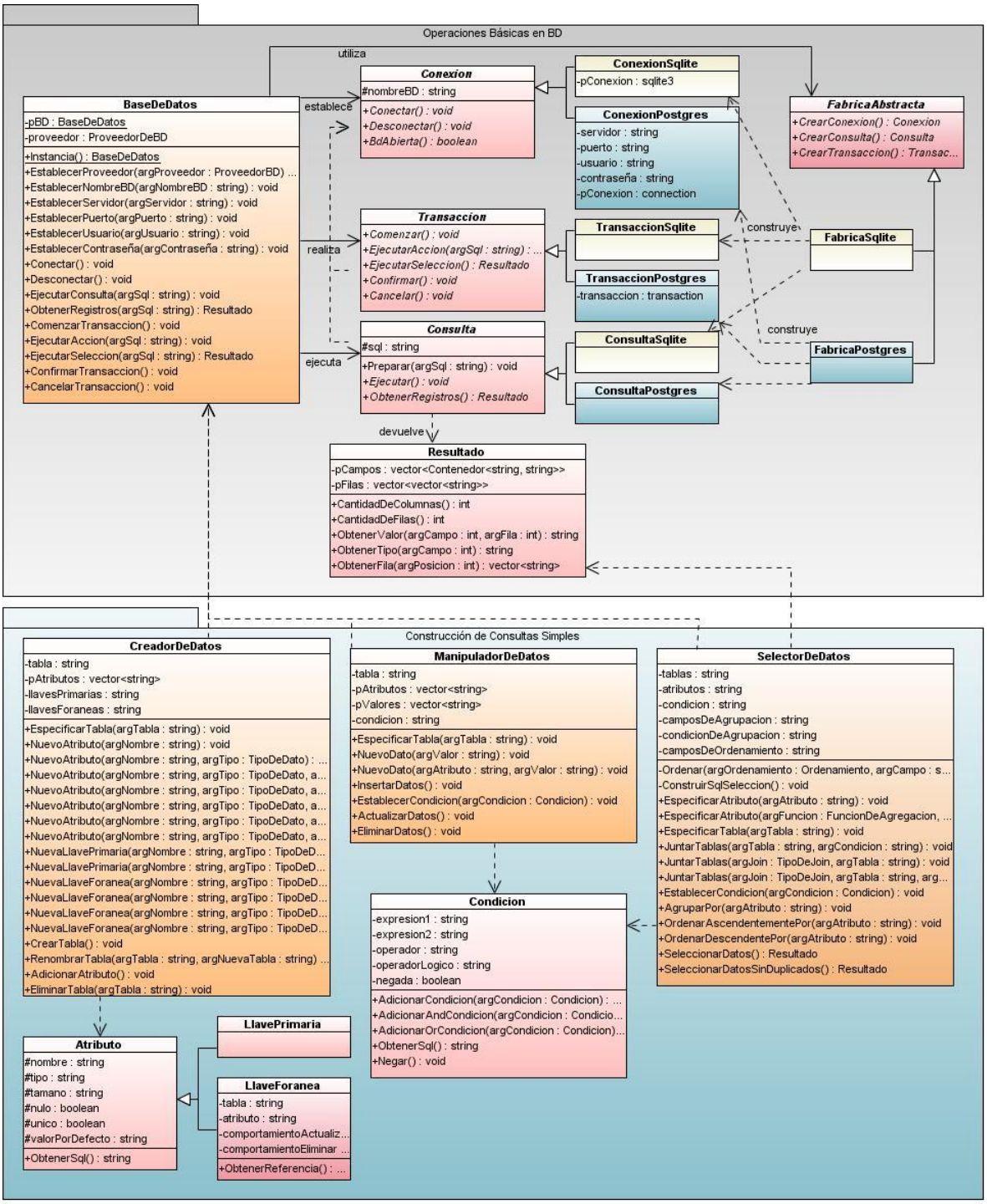

Figura 8. Diagrama de clases del diseño del componente.

4.4.1 Descripción de las clases del diseño

A continuación se describen las principales clases involucradas en el diseño del componente, haciendo énfasis en sus atributos y responsabilidades.

Tabla 6. Descripción de la clase Conexion.

Nombre: Conexion	
Atributo	**Tipo**
nombreBD	string
Responsabilidades	
Nombre:	EstablecerNombreBD(string argNombreBD)
Descripción:	Establece el nombre de la BD.
Nombre:	Conectar()
Descripción:	Establece conexión a una BD.
Nombre:	BdAbierta(): bool
Descripción:	Verifica si la conexión a la BD se encuentra abierta.
Nombre:	Desconectar()
Descripción:	Desconecta una BD.

Tabla 7. Descripción de la clase Consulta.

Nombre: Consulta	
Atributo	**Tipo**
sql	string
Responsabilidades	
Nombre:	Preparar(string argSql)
Descripción:	Prepara la consulta para ser ejecutada.
Nombre:	Ejecutar()
Descripción:	Ejecuta una consulta SQL.
Nombre:	ObtenerRegistros(): Resultado*
Descripción:	Retorna un conjunto de registros como resultado de ejecutar una consulta de selección de datos

Tabla 8. Descripción de la clase Transaccion.

Nombre: Transaccion	
Atributo	**Tipo**
Responsabilidades	
Nombre:	Comenzar()

Descripción:	Inicia una transacción.
Nombre:	Ejecutar (string argSql)
Descripción:	Ejecuta una consulta que forma parte de una transacción.
Nombre:	Obtener(string argSql): Resultado*
Descripción:	Ejecuta una consulta de selección involucrada en una transacción.
Nombre:	Confirmar()
Descripción:	Aplica los cambios provocados por una transacción en la BD.
Nombre:	Cancelar()
Descripción:	Cancela los cambios provocados por una transacción en la BD.

Tabla 9. Descripción de la clase FabricaAbstracta.

Nombre: FabricaAbstracta	
Atributo	**Tipo**
Responsabilidades	
Nombre:	CrearConexion(): Conexión*
Descripción:	Construye un puntero a un objeto Conexion.
Nombre:	CrearConsulta(): Consulta*
Descripción:	Construye un puntero a un objeto Consulta.
Nombre:	CrearTransaccion(): Transaccion*
Descripción:	Construye un puntero a un objeto Transaccion.

Tabla 10. Descripción de la clase Resultado.

Nombre: Resultado	
Atributo	**Tipo**
pCampos	vector<Contenedor<string, string>*>*
pFilas	vector<vector<string>*>*
Responsabilidades	
Nombre:	CantidadDeColumnas(): int
Descripción:	Retorna la cantidad de campos o encabezados.
Nombre:	CantidadDeFilas(): int
Descripción:	Retorna la cantidad de filas.
Nombre:	ObtenerValor(int argCampo, int argFila): string
Descripción:	Retorna el valor del campo especificado en la fila especificada.
Nombre:	ObtenerTipo(int argCampo): string
Descripción:	Retorna el tipo del campo especificado.
Nombre:	ObtenerFila(int argPosicion): vector<string>*
Descripción:	Retorna la fila que se encuentra en la posición especificada

Tabla 11. Descripción de la clase BaseDeDatos.

Nombre: BaseDeDatos	
Atributo	**Tipo**
pBD	BaseDeDatos*
proveedor	ProveedorBD
pFabrica	Fabrica*
pConexion	Conexión*
pConsulta	Consulta*
pTransaccion	Transaccion*
Responsabilidades	
Nombre:	Instancia(): BaseDeDatos*
Descripción:	Devuelve la única instancia de BD.
Nombre:	EstablecerProveedorBD(string argProveedor)
Descripción:	Establece el proveedor de BD a utilizar.
Nombre:	EstablecerNombreBD(string argNombreBD)
Descripción:	Establece el nombre de la BD.
Nombre:	EstablecerServidor(string argServidor)
Descripción:	Establece el IP del servidor de BD.
Nombre:	EstablecerPuerto(string argPuerto)
Descripción:	Establece el puerto del servidor de BD.
Nombre:	EstablecerUsuario(string argUsuario)
Descripción:	Establece el usuario para conectarse a la BD.
Nombre:	EstablecerContraseña(string argContraseña)
Descripción:	Establece la contraseña para conectarse a la BD.
Nombre:	Conectar()
Descripción:	Establece conexión a una BD.
Nombre:	Desconectar()
Descripción:	Desconecta una BD.
Nombre:	EjecutarConsulta(string argSql)
Descripción:	Ejecuta una consulta.
Nombre:	ObtenerRegistros(string argSql): Resultado*
Descripción:	Ejecuta una consulta de selección.
Nombre:	ComenzarTransaccion()
Descripción:	Inicia una transacción.
Nombre:	EjecutarAccion(string argSql)
Descripción:	Ejecuta una acción que forma parte de una transacción.
Nombre:	EjecutarSeleccion(string argSql): Resultado*
Descripción:	Ejecuta una consulta de selección que forma parte de una transacción.
Nombre:	ConfirmarTransaccion()
Descripción:	Aplica los cambios provocados por la transacción en la BD.
Nombre:	CancelarTransaccion()
Descripción:	Cancela los cambios provocados por la transacción en la BD.

Tabla 12. Descripción de la clase Atributo.

Nombre: Atributo	
Atributo	**Tipo**
nombre	string
tipo	string
tamano	int
nulo	bool
unico	bool
valorPorDefecto	string
Responsabilidades	
Nombre:	ObtenerSql(): string
Descripción:	Devuelve el segmento de código SQL que representa al atributo.

Tabla 13. Descripción de la clase Condicion.

Nombre: Condicion	
Atributo	**Tipo**
expresion1	string
expresion2	string
operador	string
negada	bool
pCondicionesAsociadas	vector<Contenedor<string, Condicion*>* >*
Responsabilidades	
Nombre:	AdicionarCondicion(Condicion* argCondicion)
Descripción:	Permite adicionar una condición.
Nombre:	AdicionarAndCondicion(Condicion* argCondicion)
Descripción:	Permite formar una condición compuesta usando el operador AND.
Nombre:	AdicionarOrCondicion(Condicion* argCondicion)
Descripción:	Permite formar una condición compuesta usando el operador OR.
Nombre:	Negar()
Descripción:	Permite negar una condición usando el operador NOT.
Nombre:	ObtenerSql(): string
Descripción:	Devuelve el segmento de código SQL que representa la condición.

Tabla 14. Descripción de la clase CreadorDeDatos.

Nombre: CreadorDeDatos	
Atributo	**Tipo**
tabla	string
pAtributos	vector<string>*
llavesPrimarias	string

llavesForaneas	string
Responsabilidades	
Nombre:	EspecificarTabla(string argTabla)
Descripción:	Permite especificar la tabla en la cual se definirán los datos.
Nombre:	NuevoAtributo(string argNombre)
Descripción:	Permite especificar el nuevo atributo a crear.
Nombre:	NuevoAtributo(string argNombre, TipoDeDato argTipo)
Descripción:	Permite especificar el nuevo atributo a crear.
Nombre:	NuevoAtributo(string argNombre, TipoDeDato argTipo, int argTamano)
Descripción:	Permite especificar el nuevo atributo a crear.
Nombre:	NuevoAtributo(string argNombre, TipoDeDato argTipo, bool argNulo, bool argUnico)
Descripción:	Permite especificar el nuevo atributo a crear.
Nombre:	NuevoAtributo(string argNombre, TipoDeDato argTipo, int argTamano, bool argNulo, bool argUnico)
Descripción:	Permite especificar el nuevo atributo a crear.
Nombre:	NuevoAtributo(string argNombre, TipoDeDato argTipo, string argValorPorDefecto)
Descripción:	Permite especificar el nuevo atributo a crear.
Nombre:	NuevoAtributo(string argNombre, TipoDeDato argTipo, int argTamano, string argValorPorDefecto)
Descripción:	Permite especificar el nuevo atributo a crear.
Nombre:	NuevaLlavePrimaria(string argNombre, TipoDeDato argTipo)
Descripción:	Permite especificar una llave primaria a crear
Nombre:	NuevaLlavePrimaria(string argNombre, TipoDeDato argTipo, int argTamano)
Descripción:	Permite especificar una llave primaria a crear
Nombre:	NuevaLlaveForanea(string argNombre, TipoDeDato argTipo, string argTabla, string argAtributo)
Descripción:	Permite especificar una llave foránea a crear
Nombre:	NuevaLlaveForanea(string argNombre, TipoDeDato argTipo, int argTamano, string argTabla, string argAtributo)
Descripción:	Permite especificar una llave foránea a crear
Nombre:	NuevaLlaveForanea(string argNombre, TipoDeDato argTipo, string argTabla, string argAtributo, ComportamientoLlavesForaneas argCompActualizar, ComportamientoLlavesForaneas argCompEliminar)
Descripción:	Permite especificar una llave foránea a crear
Nombre:	NuevaLlaveForanea(string argNombre, TipoDeDato argTipo, int argTamano, string argTabla, string argAtributo, ComportamientoLlavesForaneas argCompActualizar, ComportamientoLlavesForaneas argCompEliminar)
Descripción:	Permite especificar una llave foránea a crear
Nombre:	CrearTabla()
Descripción:	Construye y ejecuta una consulta para la creación de una tabla.
Nombre:	RenombrarTabla(string argTabla, string argNuevaTabla)
Descripción:	Permite cambiarle el nombre a una tabla.

Nombre:	AdicionarAtributo()
Descripción:	Construye y ejecuta una consulta que permite adicionar un atributo a una tabla.
Nombre:	EliminarTabla(string argTabla)
Descripción:	Construye y ejecuta una consulta que permite eliminar una tabla.

Tabla 15. Descripción de la clase ManipuladorDeDatos.

Nombre: ManipuladorDeDatos	
Atributo	**Tipo**
tabla	string
pAtributos	vector<string>*
pValores	vector<string>*
condicion	string
Responsabilidades	
Nombre:	EspecificarTabla(string argTabla)
Descripción:	Permite especificar la tabla en la cual se manipularán los datos.
Nombre:	NuevoDato(string argValor)
Descripción:	Permite especificar un nuevo dato a salvar dado el nuevo valor.
Nombre:	NuevoDato(string argAtributo, string argValor)
Descripción:	Permite especificar un nuevo dato a salvar dado el nombre del atributo y el valor.
Nombre:	InsertarDatos()
Descripción:	Construye y ejecuta una consulta de inserción de datos.
Nombre:	EstablecerCondicion(Condicion* argCondicion)
Descripción:	Establece una condición.
Nombre:	ActualizarDatos()
Descripción:	Construye y ejecuta una consulta de actualización.
Nombre:	EliminarDatos()
Descripción:	Construye y ejecuta una consulta de eliminación de datos.

Tabla 16. Descripción de la clase SelectorDeDatos.

Nombre: SelectorDeDatos	
Atributo	**Tipo**
tabla	string
atributos	string
condicion	string
camposDeAgrupacion	string
condicionDeAgrupacion	string
camposDeOrdenamiento	string
Responsabilidades	
Nombre:	EspecificarAtributo(string argAtributo)
Descripción:	Permite especificar los campos a seleccionar.

Nombre:	EspecificarAtributo(FuncionDeAgregacion argFuncion, string argAtributo)
Descripción:	Permite especificar los campos a seleccionar.
Nombre:	EspecificarTabla(string argTabla1)
Descripción:	Permite especificar la tabla.
Nombre:	JuntarTablas(string argTabla2, Condicion* argCondicion)
Descripción:	Permite combinar tablas.
Nombre:	JuntarTablas(TipoDeJoin argJoin, string argTabla2)
Descripción:	Permite combinar tablas.
Nombre:	JuntarTablas(TipoDeJoin argJoin, string argTabla2, Condicion* argCondicion)
Descripción:	Permite combinar tablas.
Nombre:	EstablecerCondicion(Condicion* argCondicion)
Descripción:	Permite establecer una condición.
Nombre:	AgruparPor(string argAtributo)
Descripción:	Permite especificar el campo de agrupación
Nombre:	OrdenarAscendentementePor(string argAtributo)
Descripción:	Permite especificar el atributo por el cual los datos serán ordenados ascendentemente.
Nombre:	OrdenarDescendentementePor(string argAtributo)
Descripción:	Permite especificar el atributo por el cual los datos serán ordenados descendentemente.
Nombre:	SeleccionarDatos(): Resultado*
Descripción:	Obtiene un conjunto de registros como resultado se ejecutar una consulta de selección de datos
Nombre:	SeleccionarDatosSinDuplicados(): Resultado*
Descripción:	Obtiene un conjunto de registros no repetidos como resultado se ejecutar una consulta de selección de datos.

4.5 Conclusiones parciales

En este capítulo se llevó a cabo el diseño del componente siguiendo los pasos propuestos por la metodología FDD. Como parte de este proceso, se adoptó la arquitectura basada en componentes teniendo en cuenta las facilidades que la misma ofrece. También se modeló la solución propuesta a través de un diagrama de clases del diseño, aplicando patrones para solucionar los problemas comunes y mejorar la calidad del diseño. Con este diagrama se proporcionó una vista global del componente, lo que permitirá una implementación eficaz de las clases.

Capítulo 5. Construcción y pruebas

5.1 Introducción

En este capítulo se presenta el estilo de codificación utilizado durante la implementación del componente. También se diseñan pruebas de unidad para sus principales clases haciendo uso de la técnica pruebas de caja blanca. Finalmente, se muestran los resultados de las pruebas a las que fue sometida la aplicación, con vistas a corregir las deficiencias encontradas y verificar su correcto funcionamiento.

5.2 Estilo de codificación

Un estilo o estándar de codificación es el conjunto de reglas o normas usadas para escribir código, incluyendo numerosos aspectos dentro del proceso de codificación. La utilización de estilos de codificación facilita la lectura, comprensión y mantenimiento del código, y contribuye a la calidad del mismo. También favorece la rápida corrección de los errores y la comunicación entre los miembros del equipo de desarrollo.

Existen diferentes estilos de capitalización para los identificadores (Fosch Alonso and Arellano Roig 2006):

- *Estilo Pascal*: Se capitaliza la primera letra de cada palabra.
- *Estilo Camello*: Se capitaliza la primera letra de cada palabra, excepto en la primera palabra.
- *Estilo Versal o Mayúsculas*: Todo se escribe en mayúsculas.

Para la implementación del componente se empleó el siguiente estilo de código:

Identificadores:

- Las clases serán nombradas con sustantivos usando el estilo Pascal y sin utilizar prefijos.
- Las interfaces serán nombradas utilizando el estilo Pascal. Se colocará como prefijo "I".
- Los métodos serán nombrados con verbos o frases verbales usando el estilo Pascal.
- Los atributos, las variables y los parámetros serán nombrados usando el estilo Camello. Sus nombres deben ser descriptivos. Los contadores en ciclos utilizarán como nombre las letras *i*, *j* y *k*.

- Las constantes serán declaradas en mayúsculas. En caso de estar compuestas por varias palabras, se utilizará el caracter "_" como separador entre las mismas.
- Los valores de los enumerados serán nombrados en singular utilizando el estilo Versal.

Comentarios: Para los comentarios de implementación y documentación se utilizarán los formatos // y /* */, respectivamente. En las declaraciones de las clases se incluirá el siguiente comentario:

```
/**************************************************************
 Clase: Nombre de la clase.
 Descripción: Descripción de la clase.
 Autor: Autor de la clase.
 Fecha: Fecha de creación de la clase.
**************************************************************/
```

Figura 9. Comentario de las declaraciones de clases.

Identación: La identación de los bloques de sentencias tendrá una longitud de 4 espacios.
Idioma: Todas las declaraciones serán escritas en idioma Español.

```
void CreadorDeDatos::AdicionarAtributo()
{
    if(this->tabla == "")
        throw new Excepcion("Nombre de la tabla no especificado.");
    if(this->pAtributos->size() == 0)
        throw new Excepcion("Atributo no especificado.");

    for(int unsigned i = 0; i < this->pAtributos->size(); i++)
    {
        string sql = "ALTER TABLE ";
        sql.append(this->tabla);
        sql.append(" ADD COLUMN ");
        sql.append(this->pAtributos->at(i));
        sql.append(";");

        BaseDeDatos::EjecutarConsulta(sql);
    }
    this->Limpiar();
}
```

Figura 10. Segmento de código con el estilo aplicado.

5.3 Pruebas de unidad

Las pruebas son una actividad que persigue verificar el correcto funcionamiento de un software, así como garantizar la calidad del mismo. También permiten asegurar que el software cumple con la funcionalidad para la cual fue diseñado. Consisten en la ejecución de un sistema o componente bajo determinadas condiciones, con vistas a detectar las deficiencias y errores que presenta. De esta forma, los errores encontrados pueden ser corregidos antes de entregar el producto a los clientes. Debido a su importancia, las pruebas deben tenerse en cuenta a lo largo de todo el ciclo de vida del producto.

Existen numerosos tipos de pruebas, entre las que se encuentra las pruebas de unidad. Estas pruebas se enfocan a unidades pequeñas de software, las cuales pueden ser clases o componentes. Las mismas permiten verificar los flujos de control y de datos, así como la correcta codificación de los componentes o clases. Las pruebas de clases están dirigidas por las operaciones que estas encapsulan. Estas operaciones se prueban como parte de la clase a la que pertenecen y no de forma aislada. Las pruebas de unidad utilizan de forma intensiva las técnicas de prueba de caja blanca (Natalia Juristo and Moreno 2006).

5.3.1 Técnica de pruebas de caja blanca

La técnica de pruebas de caja blanca constituye uno de los tipos de pruebas más importantes que se le aplican a los sistemas informáticos, tributando a la disminución de los errores existentes en los mismos. Permite examinar el comportamiento y la estructura del código escrito. Tiene como objetivo diseñar casos de prueba de modo que todas las sentencias del programa se ejecuten al menos una vez. Para garantizar este objetivo se aplica la prueba del camino básico, la cual se basa en obtener una medida de la complejidad del diseño procedimental de un programa. Este valor representa el número máximo de casos de prueba que deben realizarse y se conoce como complejidad ciclomática (Natalia Juristo and Moreno 2006).

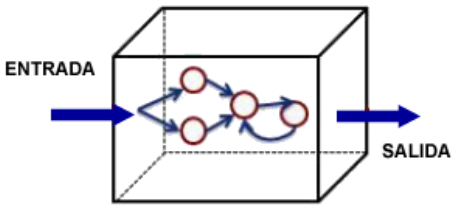

Figura 11. Representación de las pruebas de caja blanca.

5.3.2 Casos de prueba

A continuación se presentan los casos de prueba definidos para las principales clases del componente.

A. Caso de prueba “Conexión a una BD SQLite”.

Esta prueba se aplica al método Conectar perteneciente a la clase ConexionSqlite:

```
void ConexionSqlite::Conectar()                                          
{                                                                        
    if(this->nombreBD == "")                                             1
        throw new Excepcion("Nombre de BD no especificado.");            2
                                                                         
    int rc = sqlite3_open(this->nombreBD.c_str(), &this->pBD);           3
    if(rc != SQLITE_OK)                                                  4
        throw new Excepcion("No se pudo establecer conexión a la BD.");  5
}                                                                        6
```

Figura 12. Numeración de las sentencias del método Conectar de la clase ConexionSqlite.

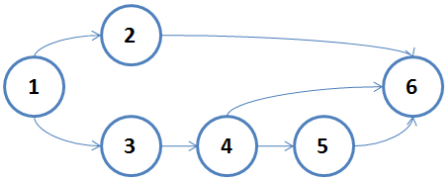

Figura 13. Grafo del método Conectar de la clase ConexionSqlite.

Cálculo de la complejidad ciclomática

V (G): Complejidad ciclomática.
V (G) = A - N + 2 donde: *A* es el número de aristas del grafo y *N* es el número de nodos.
V (*G*) = *P* + 1 donde: *P* es el número de nodos predicado contenidos en el grafo *G*.
V (*G*) = 3

Camino 1: 1-2-6.
Camino 2: 1-2-4-6.
Camino 3: 1-3-4-5-6.

Tabla 17. Caso de prueba "Conexión a una BD SQLite".

Número del camino	Caso de prueba	Objetivo	Resultado esperado	Resultado obtenido
1	nombreBD = "".	Probar la conexión a una BD SQLite sin haber especificado su nombre.	Lanzamiento de la excepción "Nombre de BD no especificado".	Se lanzó la excepción "Nombre de BD no especificado".
2	nombreBD = "/home/milenis/Escritorio/CABD/bd.db"	Probar la conexión a una BD SQLite.	Si la BD existe se establece conexión a la misma. En caso contrario, se crea una nueva BD con el nombre especificado y se establece conexión a la misma.	Se estableció conexión a la BD especificada.
3	nombreBD = "/home/milenis/Escritorio/CABD/otrabd.db"	Probar la conexión a una BD SQLite a la que no se tienen permisos de lectura ni de escritura.	Lanzamiento de la excepción "No se pudo establecer conexión a la BD".	Se lanzó la excepción "No se pudo establecer conexión a la BD".

B. Caso de prueba "Ejecución de consultas de selección en una BD SQLite".

Se aplica al método ObtenerRegistros perteneciente a la clase ConsultaSqlite:

```
Resultado* ConsultaSqlite::ObtenerRegistros()
{
    Resultado* registros = new Resultado();                                                    1
    Conexion* pObjConexion = BaseDeDatos::Instancia()->ObtenerConexion();                      1
    sqlite3* conexion = dynamic_cast<ConexionSqlite*>(pObjConexion)->ObtenerConexion();        1
    sqlite3_stmt* resultado;                                                                   1
    const char* siguiente;                                                                     1
    int msg = sqlite3_prepare(conexion, this->sql.c_str(), sql.length(), &resultado, &siguiente); 1

    if(msg != SQLITE_OK)                                                                       2
        throw new Excepcion("Error en la consulta.");                                          3
    else
    {
        registros = this->AlmacenarResultado(resultado);                                       4
        sqlite3_finalize(resultado);                                                          15
        return registros;                                                                     15
    }
}                                                                                             16
          Resultado* ConsultaSqlite::AlmacenarResultado(sqlite3_stmt* argResultado)
          {
              int msg = sqlite3_step(argResultado);                                           5
              Resultado* registros = new Resultado();                                         5
              int ncolumnas = sqlite3_column_count(argResultado);                             5

              if(ncolumnas > 0)                                                               6
              {
                  vector<string>* fila = new vector<string>();                                7
                  string tipoDeColumna = "";                                                  7
                  string nombreDeColumna = "";                                                7
                  string valorDeColumna = "";                                                 7
                  char* valor = NULL;                                                         7

                  //Obteniendo los tipos y nombres de columnas
                  for(int i = 0; i < ncolumnas; i++)                                          8
                  {
                      tipoDeColumna = (string)sqlite3_column_decltype(argResultado, i);       9
                      nombreDeColumna = (string)sqlite3_column_name(argResultado, i);         9
                      registros->InsertarCampo(nombreDeColumna, tipoDeColumna);               9
                  }
```

```
        //Obteniendo los tipos y nombres de columnas
        for(int i = 0; i < ncolumnas; i++)                                    8
        {
            tipoDeColumna = (string)sqlite3_column_decltype(argResultado, i); 9
            nombreDeColumna = (string)sqlite3_column_name(argResultado, i);   9
            registros->InsertarCampo(nombreDeColumna, tipoDeColumna);         9
        }

        //Obteniendo las filas con sus valores
        while(msg == SQLITE_ROW)                                             10
        {
            for(int i = 0; i < ncolumnas; i++)                               11
            {
                valor = (char*)sqlite3_column_text(argResultado, i);         12
                valorDeColumna = (string)valor;                              12
                fila->push_back(valor);                                      12
            }
            registros->InsertarFila(fila);                                   13
            fila = new vector<string>();                                     13
            msg = sqlite3_step(argResultado);                                13
        }
    }
    return registros;                                                        14
}
```

Figura 14. Numeración de las sentencias del método ObtenerRegistros de la clase ConsultaSqlite.

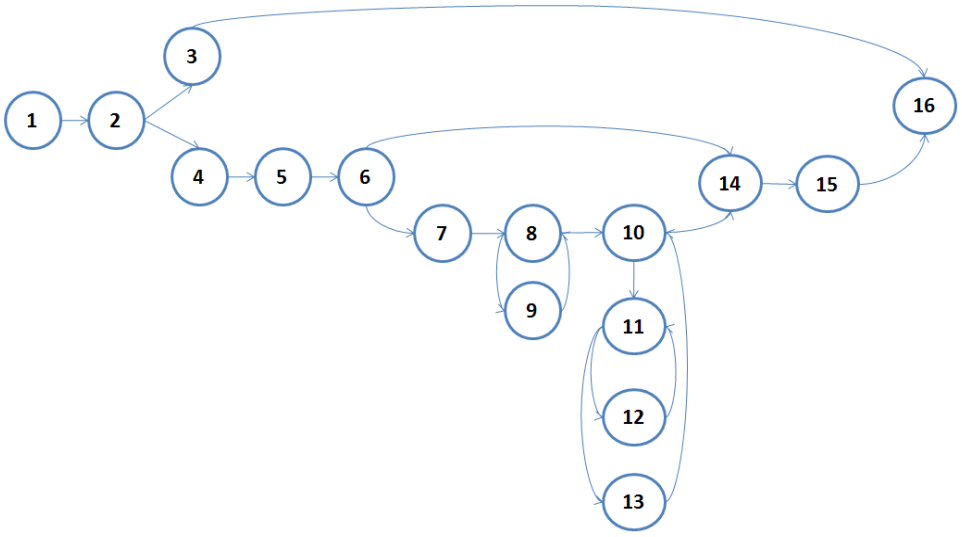

Figura 15. Grafo del método ObtenerRegistros de la clase ConsultaSqlite.

Cálculo de la complejidad ciclomática

V (G) = 6

Camino 1: 1-2-3-16.

Camino 2: 1-2-4-5-6-14-15-16.

Camino 3: 1-2-4-5-6-7-8-10-14-16.

Camino 4: 1-2-4-5-6-7-8-9-8-10-16-15-16.

Camino 5: 1-2-4-5-6-7-8-9-8-10-11-13-10-14-15-16.

Camino 6: 1-2-4-5-6-7-8-9-8-10-11-12-11-13-10-14-15-16.

Tabla 18. Caso de prueba "Ejecución de consultas de selección en una BD SQLite".

Número del camino	Caso de prueba	Objetivo	Resultado esperado	Resultado obtenido
1	Conexión establecida. sql = "SELECT FROM collars".	Probar la ejecución de consultas de selección erróneas en una BD SQLite.	Lanzamiento de la excepción "Error en la consulta".	Se lanzó la excepción "Error en la consulta".
2	Conexión establecida. sql = "SELECT * FROM collars".	Probar la ejecución de consultas de selección en una BD SQLite, las cuales no retornan datos.	Se espera que la consulta se ejecute exitosamente.	La consulta se ejecutó exitosamente.
3	No es posible realizar este camino.			
4	No es posible realizar este camino.			
5	No es posible realizar este camino.			
6	Conexión establecida. sql = "SELECT * FROM collars".	Probar la ejecución de consultas de selección en una BD SQLite.	Se espera que la consulta se ejecute exitosamente.	La consulta se ejecutó exitosamente.

C. Caso de prueba "Construcción de consultas de inserción de datos para una BD SQLite".

Se aplica al método InsertarDatos perteneciente a la clase ManipuladorDeDatos:

```
void ManipuladorDeDatos::InsertarDatos()
{
    if(this->tabla == "")                                                    1
        throw new Excepcion("Nombre de tabla no especificado.");             2
    if(this->pValores->size() == 0)                                          3
        throw new Excepcion("Valores a insertar no especificados.");         4

    string sql = "INSERT INTO " + this->tabla;                               5
    if(this->pAtributos->size() > 0)                                         6
    {
        sql.append(" (");                                                    7
        for(int unsigned i = 0; i < this->pAtributos->size(); i++)           8
        {
            if(i != 0)                                                       9
                sql.append(", ");                                           10
            sql.append(this->pAtributos->at(i));                            11
        }
        sql.append(")");                                                    12
    }

    sql.append(" VALUES (");                                                13
    for(int unsigned j = 0; j < this->pValores->size(); j++)                14
    {
        if(j != 0)                                                          15
            sql.append(", ");                                               16
        sql.append("'");                                                    17
        sql.append(this->pValores->at(j));                                  17
        sql.append("'");                                                    17
    }
    sql.append(");");                                                       18
    BaseDeDatos::Instancia()->EjecutarConsulta(sql);                        18
    this->Limpiar();                                                        23
}                                                                           25
```

```
void BaseDeDatos::EjecutarConsulta(string argSql)
{
    if(Instancia()->pConexion->BdAbierta())                    19
    {
        Instancia()->pConsulta->Preparar(argSql);              23
        Instancia()->pConsulta->Ejecutar();                    23
    }
    else
        throw new Excepcion("Conexión no establecida a la BD."); 24
}
```

```
bool ConexionSqlite::BdAbierta()
{
    if(this->seConecto == SQLITE_OK)  19
        return true;                  20
    else
        return false;                 24
}
```

```
void ConsultaSqlite::Ejecutar()
{
    Conexion* pObjConexion = BaseDeDatos::Instancia()->ObtenerConexion();                20
    sqlite3* conexion = dynamic_cast<ConexionSqlite*>(pObjConexion)->ObtenerConexion();  20
    char* error;                                                                         20
    int rc = sqlite3_exec(conexion,this->sql.c_str(),NULL,NULL,&error);                  20

    if(rc != SQLITE_OK)                                                                  21
    {
        sqlite3_free(error);                                                             22
        throw new Excepcion(error);                                                      22
    }
}
```

Figura 16. Numeración de las sentencias del método InsertarDatos de la clase ManipuladorDeDatos.

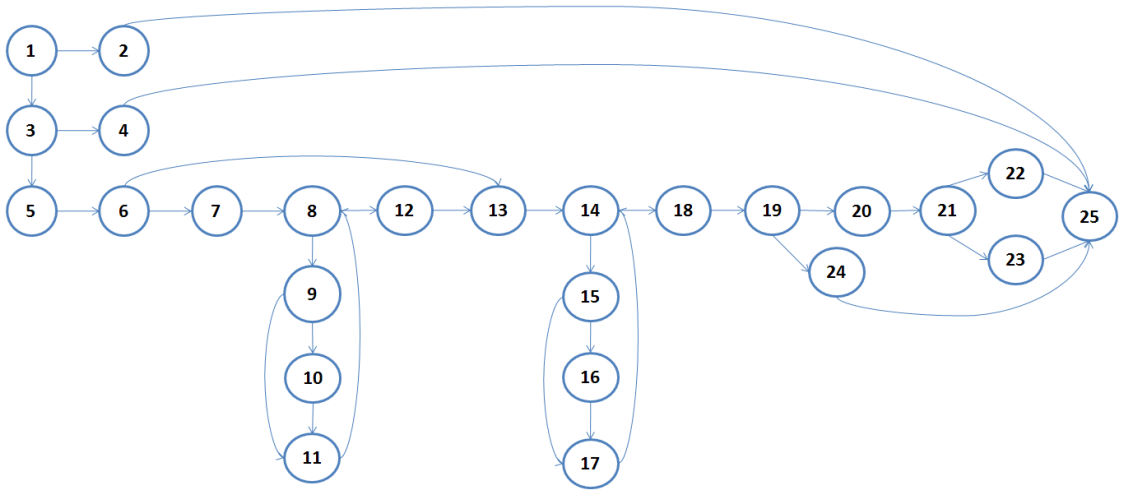

Figura 17. Grafo del método InsertarDatos de la clase ManipuladorDeDatos.

Cálculo de la complejidad ciclomática

V (G) = 10

Camino 1: 1-2-25.

Camino 2: 1-3-4-25.

Camino 3: 1-3-5-6-13-14-15-16-17-14-18-19-24-25.

Camino 4: 1-3-5-6-13-14-15-16-17-14-18-19-20-21-22-25.

Camino 5: 1-3-5-6-13-14-18-19-20-21-23-25.

Camino 6: 1-3-5-6-7-8-12-13-14-18-19-20-21-23-25.

Camino 7: 1-3-5-6-7-8-9-11-12-13-14-18-19-20-21-23-25.

Camino 8: 1-3-5-6-7-8-9-10-11-12-13-14-18-19-20-21-23-25.

Camino 9: 1-3-5-6-13-14-15-16-17-18-19-20-21-23-25.

Camino 10: 1-3-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-23-25.

Tabla 19. Caso de prueba "Construcción de consultas de inserción de datos".

Número del camino	Caso de prueba	Objetivo	Resultado esperado	Resultado obtenido
1	tabla = "" pAtributos = [] pValores = []	Probar la inserción de datos sin especificar la tabla en cual deberán insertarse.	Lanzamiento de la excepción "Nombre de tabla no especificado".	Se lanzó la excepción "Nombre de tabla no especificado".
2	tabla = tabla01 pAtributos = [] pValores = []	Probar la inserción de datos sin especificar los valores a insertar.	Lanzamiento de la excepción "Valores a insertar no especificados".	Se lanzó la excepción "Valores a insertar no especificados".
3	tabla = tabla01 pAtributos = [] pValores = [1, texto1]	Probar la inserción de datos en una BD a la que no se ha establecido conexión.	Lanzamiento de la excepción "Conexión no establecida a la BD".	Se lanzó la excepción "Conexión no establecida a la BD".
4	tabla = tablanoexiste pAtributos = [] pValores = [1, texto1]	Probar la inserción de datos en una tabla que no existe.	Lanzamiento de la excepción "Error en la consulta".	Se lanzó la excepción "Error en la consulta".
5	No es posible realizar este camino.			
6	No es posible realizar este camino.			
7	No es posible realizar este camino.			
8	No es posible realizar este camino.			
9	tabla = tabla01 pAtributos = [] pValores = [1, texto1]	Probar la inserción de datos en una tabla, sin especificar los atributos a insertar.	Se espera la inserción de los datos en la tabla especificada.	Los datos fueron insertados en la tabla especificada.
10	tabla = tabla01 pAtributos = [atributo01,	Probar la inserción de datos en una tabla.	Se espera la inserción de los datos en la	Los datos fueron insertados en la tabla

	atributo02] pValores = [1, texto1]		tabla especificada.	especificada.

5.4 Conclusiones parciales

En este capítulo se definió el estilo de codificación utilizado, el cual facilitará la compresión y mantenimiento del código. También se probaron de forma individual las clases que conforman el componente, utilizando la técnica pruebas de caja blanca. La aplicación de estas pruebas permitió rectificar los errores y deficiencias fortaleciendo la calidad del producto. Además, permitieron comprobar que el componente realiza correctamente las funcionalidades para las cuales fue diseñado.

Conclusiones generales

El desarrollo de la presente solución informática para la abstracción de BD del sistema GeolMin, permitió dar cumplimiento al objetivo trazado al inicio de la investigación; con la cual se pudo arribar a las siguientes conclusiones:

- El componente desarrollado permite manipular la información del sistema GeolMin con independencia del tipo de BD utilizada, facilitando la sustitución de una tecnología de BD por otra.
- El componente desarrollado brinda soporte a los sistemas gestores SQLite y PostgreSQL, siendo fácilmente extensible a otros gestores.
- El componente desarrollado facilita la labor de los programadores en las tareas relativas a la implementación del acceso a los datos, permitiéndoles abstraerse del SGBD utilizado. Además, les permite utilizar una sintaxis única e independiente del SGBD para acceder y manipular los datos.
- El componente desarrollado cumple con las exigencias impuestas por los clientes ya que realiza de forma correcta los procesos de conexión, ejecución de consultas y realización de transacciones; lo cual se comprobó mediante la aplicación de pruebas unitarias a nivel de clases.
- Constituye un componente reutilizable por distintas aplicaciones informáticas, lo que permitirá ahorrar tiempo y esfuerzo durante el futuro desarrollo de soluciones informáticas.

Recomendaciones

En las próximas versiones del componente se recomienda:

- Extender sus funcionalidades de modo que brinde soporte a otros sistemas gestores y permita la conexión a varias BD de manera simultánea.
- Extender las funcionalidades correspondientes a la construcción de consultas SQL, permitiendo la construcción de consultas de mayor complejidad.
- Extender el manejo de excepciones haciendo uso de un sistema de excepciones específicas que garanticen una adecuada gestión de los errores.

Bibliografía referenciada

Amaro Calderón, S. D. and J. C. Valverde Rebaza (2007). Metodologías ágiles. Perú.

Army_Net_Centric_Data. (2009). "Army Net-Centric Data Strategy." from http://data.army.mil/.

Calabria, L. (2003). Metodología FDD. Uruguay.

Camallea, N. L. N. (2004). Gestión de Base de Datos con ADO.NET. Ciudad de La Habana, Científico Técnica.

Cooper, J. W. (1998). The Design Patterns Java Companion, Addison Wesley.

Date, C. J. (2003). Introducción a los sistemas de bases de datos. La Habana, Félix Varela.

De Miguel Castaño, A., M. Piattini Velthuis, et al. (1999). Diseño de bases de datos relacionales. México, RAMA.

Douglas, K. and S. Douglas (2003). PostgreSQL.

Ezequiel Rozic, S. (2004). Bases de Datos. Buenos Aires, MP Ediciones.

Fosch Alonso, I. and J. Arellano Roig (2006). Guía de estilo de codificación.

Hernández Orallo, E. (2002). El Lenguaje Unificado de Modelado.

IBM. (2011). "Sitio oficial de IBM. Rational Rose Enterprise Edition." from http://www-142.ibm.com/software/products/es/es/enterprise/.

Jacobson, I., G. Booch, et al. (2000). El Proceso Unificado de Desarrollo de Software. Madrid, Pearson Educación S.A.

Joshi, B., P. Dickinson, et al. (2001). Professional ADO.NET, Wrox Press.

Katrib Mora, M. (1997). Programación orientada a objeto en C++.

Larman, C. (2002). UML y patrones. Introducción al análisis y diseño orientado a objetos. México, Prentice Hal.

Martínez, R. (2010). "Portal en español de PostgreSQL." from http://www.postgresql-es.org/sobre_postgresql.

Mato García, R. M. (1999). Diseño de Bases de Datos.

Matthew, N. and R. Stones (2005). Beginning Databases with PostgreSQL: From Novice to Professional. New York.

Merriam_Webster. (2011). "Enciclopedia Británica: Merriam-Webster." from http://www.merriam-webster.com.

Bibliografía referenciada

Microsoft. (2011). "Microsoft Open Database Connectivity (ODBC)." from http://msdn.microsoft.com/en.../ms710252(v=vs.85).aspx.

Microsoft. (2011). "OLE DB Provider Templates (C++)." from http://msdn.microsoft.com/en-us/library/h63swas7(v=vs.80).aspx.

Microsoft. (2011). "Sitio oficial de Microsoft. Visual C++." from http://msdn.microsoft.com/es-es/library/60k1461a.aspx.

Moreno, F. (2000). Introducción a la POO, Grupo EIDOS.

Natalia Juristo, A. M. and S. V. Moreno (2006). Técnicas de evaluación de software.

Nguèn, M. H. (2008). Data Abstraction Layer: Independet for Database.

Nokia. (2011). "Sitio oficial de Qt. Qt development tools." from http://qt.nokia.com/products/developer-tools/.

Otto, A. and D. Hinderink (2004). TYPO3 Database Abstraction.

Owens, M. (2006). The Definitive Guide to SQLite. New York, Apress.

Parker, D. A. and M. Hoenicka (2005). Database Independent Abstraction Layer for C: libdbi Programmer´s Guide.

Peláez, J. C. (2009). "Blog de Juan Peláez en Geeks.ms. Arquitectura basada en componentes." from http://geeks.ms/blogs/jkpelaez/archive/2009/04/18/arquitectura-basada-en-componentes.aspx.

Pérez Valls, I. (2009). "JavaDabbaDoo.org. Introducción a los patrones." from http://www.javadabbadoo.org/.../paso03patrones.html.

PGDG. (2011). "Sitio oficial de PostgreSQL." from http://www.postgresql.org/about/advantages.

Pressman, R. S. (2005). Ingeniería de Software. Un enfoque práctico, McGraw Hill Higher Education.

RAE. (2011). "Diccionario de la Real Academia Española." Abstraer, from http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=abstraer.

Rodríguez, G. J. S. (2008). Evolución de los lenguajes de programación ¿Por qué cambiarse a la Programación Orientada a Objetos?

Thelin, J. (2007). Foundations of QT Development.

Visual_Paradigm_International (2011). Sitio oficial de Visual Paradigm. Visual Paradigm for UML - UML tool for software application development.

Yunta, L. R. (2001). Bases de datos documentales: su estructura y uso. Madrid, CINDOC.

Zator_Systems (2011). Curso C++. El lenguaje C++.

Bibliografía consultada

Amaro Calderón, S. D. and J. C. Valverde Rebaza (2007). Metodologías ágiles. Perú.

ANSI. (2011). "Sitio oficial de ANSI." from http://www.ansi.org.

Army_Net_Centric_Data. (2009). "Army Net-Centric Data Strategy." from http://data.army.mil/.

Calabria, L. (2003). Metodología FDD. Uruguay.

Camallea, N. L. N. (2004). Gestión de Base de Datos con ADO.NET. Ciudad de La Habana, Científico Técnica.

Cooper, J. W. (1998). The Design Patterns Java Companion, Addison Wesley.

Cuadra, D., E. Castro, et al. Diseño de Bases de Datos, CNICE (Centro Nacional de Información y Comunicación Educativa).

Date, C. J. (2003). Introducción a los sistemas de bases de datos. La Habana, Félix Varela.

De Miguel Castaño, A., M. Piattini Velthuis, et al. (1999). Diseño de bases de datos relacionales. México, RAMA.

Ezequiel Rozic, S. (2004). Bases de Datos. Buenos Aires, MP Ediciones.

Fosch Alonso, I. and J. Arellano Roig (2006). Guía de estilo de codificación.

Hernández Orallo, E. (2002). El Lenguaje Unificado de Modelado.

IBM. (2011). "Sitio oficial de IBM. Rational Rose Enterprise Edition." from http://www-142.ibm.com/software/products/es/es/enterprise/.

Jacobson, I., G. Booch, et al. (2000). El Proceso Unificado de Desarrollo de Software. Madrid, Pearson Educación S.A.

John Metsker, S. (2002). Design Patterns Java Workbook, Addison Wesley.

Joshi, B., P. Dickinson, et al. (2001). Professional ADO.NET, Wrox Press.

Larman, C. (2002). UML y patrones. Introducción al análisis y diseño orientado a objetos. México, Prentice Hal.

Marcelo Hernán, S. (2004). Diseño de una Metodología Ágil de Desarrollo de Software. Buenos Aires, FIUBA (Universidad de Buenos Aires). **Ingeniería en Informática**.

Mato García, R. M. (1999). Diseño de Bases de Datos.

Merriam_Webster. (2011). "Enciclopedia Británica: Merriam-Webster." from http://www.merriam-webster.com.

Bibliografía consultada

Microsoft. (2011). "Microsoft Open Database Connectivity (ODBC)." from http://msdn.microsoft.com/en.../ms710252(v=vs.85).aspx.

Microsoft. (2011). "OLE DB Provider Templates (C++)." from http://msdn.microsoft.com/en-us/library/h63swas7(v=vs.80).aspx.

Microsoft (2011). Sitio oficial de Microsoft. Microsoft Open Database Connectivity.

Ministerio_de_la_Industria_Básica. (2001). "Ministerio de la Industria Básica." from www.cubagob.cu/des_eco/minbas.htm.

Moreno, F. (2000). Introducción a la POO, Grupo EIDOS.

Natalia Juristo, A. M. and S. V. Moreno (2006). Técnicas de evaluación de software.

Nguèn, M. H. (2008). Data Abstraction Layer: Independet for Database.

Nokia. (2011). "Sitio oficial de Qt. Qt development tools." from http://qt.nokia.com/products/developer-tools/.

OMG. (2011). "Object Management Group. Business Process Management Initiative." from http://www.bpmn.org/index.htm.

OMG. (2011). "Sitio oficial de OMG SysML." OMG Systems Modeling Language, from http://www.omgsysml.org/.

OMG. (2011). "Unified Modeling Language." from http://www.uml.org/.

Otto, A. and D. Hinderink (2004). TYPO3 Database Abstraction.

Owens, M. (2006). The Definitive Guide to SQLite. New York, Apress.

Parker, D. A. and M. Hoenicka (2005). Database Independent Abstraction Layer for C: libdbi Programmer´s Guide.

Peláez, J. C. (2009). "Blog de Juan Peláez en Geeks.ms. Arquitectura basada en componentes." from http://geeks.ms/blogs/jkpelaez/archive/2009/04/18/arquitectura-basada-en-componentes.aspx.

Pérez Valls, I. (2009). "JavaDabbaDoo.org. Introducción a los patrones." from http://www.javadabbadoo.org/.../paso03patrones.html.

Pressman, R. S. (2005). Ingeniería de Software. Un enfoque práctico, McGraw Hill Higher Education.

RAE. (2011). "Diccionario de la Real Academia Española." Abstraer, from http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=abstraer.

Rodríguez, G. J. S. (2008). Evolución de los lenguajes de programación ¿Por qué cambiarse a la Programación Orientada a Objetos?

Bibliografía consultada

Thelin, J. (2007). Foundations of QT Development.

Visual_Paradigm_International (2011). Sitio oficial de Visual Paradigm. Visual Paradigm for UML - UML tool for software application development.

Yunta, L. R. (2001). Bases de datos documentales: su estructura y uso. Madrid, CINDOC.

Zator_Systems (2011). Curso C++. El lenguaje C++.

Glosario de términos

Aislamiento: Propiedad que garantiza que los efectos provocados por una transacción se encuentren aislados de los efectos que provocan otras transacciones concurrentes.

ANSI (*American National Standards International*): Organización privada sin fines de lucro que supervisa el desarrollo de estándares para productos, servicios y procesos en los Estados Unidos.

API (*Applications Programming Interface*, Interfaz de Programación de Aplicaciones): Interfaz que permite la comunicación e interacción entre componentes de software, a través de un conjunto de funciones y procedimientos.

Atomicidad: Propiedad que garantiza que todas las modificaciones realizadas sobre los datos de una transacción, se completen como un grupo si la transacción tiene éxito, o se cancelen si falla.

Base de datos objeto - relacional: Base de datos relacional a la cual se le incorporan conceptos del paradigma orientado a objetos.

Componente: Unidad de software independiente y reemplazable que encapsula un conjunto de servicios o funciones relacionadas a través de una interfaz.

Consistencia: Propiedad que preserva la integridad de los datos al finalizar una transacción, ya sea exitosa o fallidamente.

Driver de base de datos: Programa informático que permite a las aplicaciones interactuar con un Sistema Gestor de Base de Datos específico.

Durabilidad: Esta propiedad garantiza que una vez confirmada una transacción sus efectos sean permanentes en la base de datos.

Encapsulamiento: Empaquetamiento de los atributos y métodos dentro de un objeto. Ocultamiento de los miembros de un objeto, de modo que solo pueden ser modificados por las operaciones definidas para ese objeto.

EULA (*End User License Agreement,* Acuerdo de licencia para el usuario final): Licencia que restringe el uso de un producto a un único usuario, el comprador.

Interfaz: Colección de operaciones que son utilizadas para especificar los servicios o funciones que ofrece un componente.

Modelo cliente-servidor: Modelo en el cual un cliente realiza peticiones a un servidor encargado de satisfacer las mismas.

OMG (*Object Management Group*): Consorcio de especificaciones de la industria informática (sin fines de lucro), que define y mantiene las especificaciones de UML.