Universidad de las Ciencias Informáticas

Facultad 6

Componente para el mapeo objeto-relacional en el sistema Syam

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

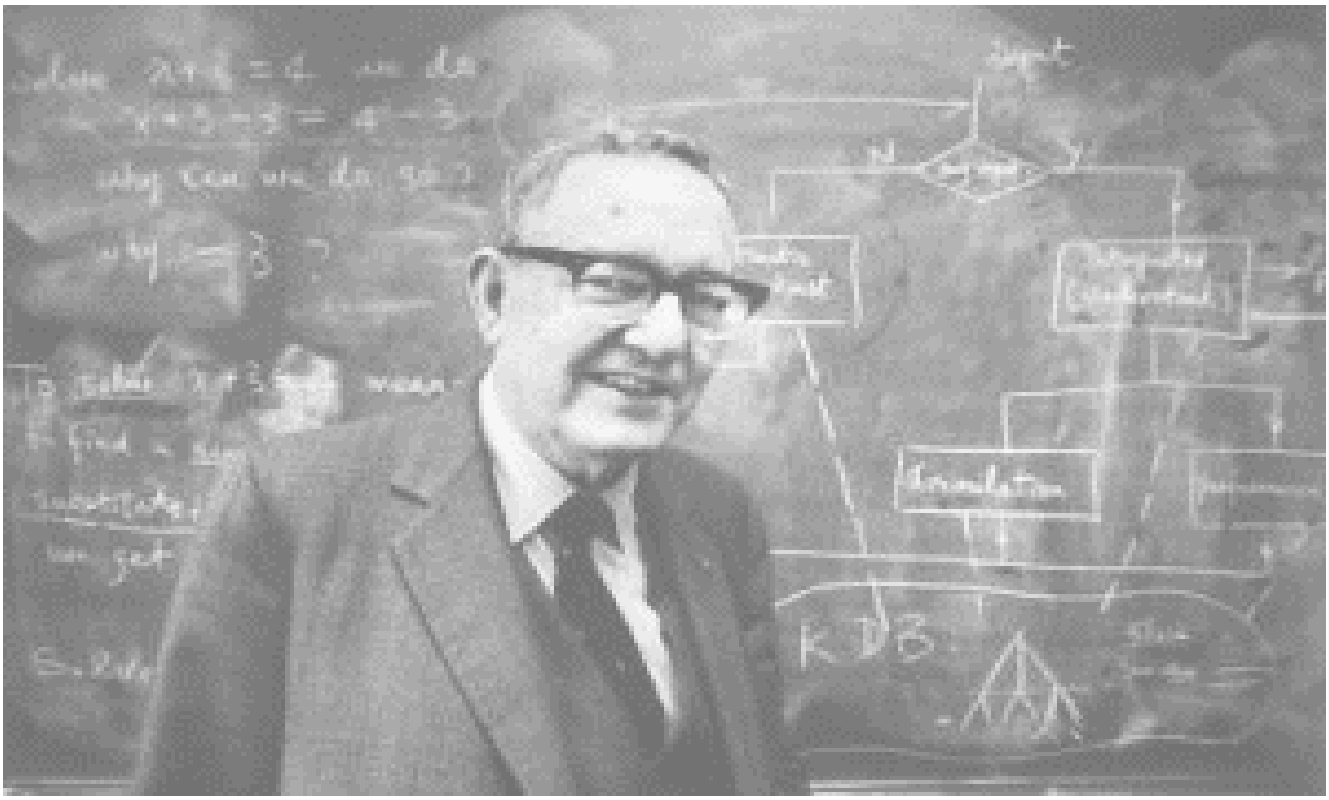
Autores: Jorge Luis Lezcano Díaz

Luis Daniel Valdés Román

Tutora: Ing. Adisley Reyes Crespo

Ing. Indira Guerra Rodríguez

La Habana, mayo de 2015



No hay moral en torno a la tecnología. La tecnología expande nuestras formas de pensar sobre las cosas, amplía nuestras formas de hacer las cosas.

Herbert Simon

Declaración de autoría

Declaramos por este medio que nosotros, Jorge Luis Lezcano Díaz y Luis Daniel Valdés Román, somos los autores de este trabajo y que autorizamos a la Universidad de las Ciencias Informáticas para hacer uso del mismo en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste, se firma la presente declaración jurada de autoría en La Habana a los ____ días del mes _____ del año _____.

Jorge Luis Lezcano Díaz
Autor

Luis Daniel Valdés Román
Autor

Ing. Adisley Reyes Crespo
Tutor

Ing. Indira Guerra Rodríguez
Tutor

Datos de Contacto

Tutor:

Ing. Adisley Reyes Crespo: Ingeniera en Ciencias Informáticas, es profesora asistente. Cuenta con 8 años de experiencia. Se graduó en el año 2007 en la Universidad de las Ciencias Informáticas.

Correo electrónico: areyesc@uci.cu

Tutor:

Ing. Indira Guerra Rodríguez: Ingeniera en Ciencias Informáticas. Cuenta con 2 años de experiencia. Se graduó en el año 2013 en la Universidad de las Ciencias Informáticas.

Correo electrónico: igrodriguez@uci.cu

Agradecimientos

Luis Daniel:

A mis padres Ivonne y Ángel Luis, nunca podré agradecerles lo suficiente, es por ustedes que estoy aquí hoy, soy el hombre que soy por tenerlos a ustedes como ejemplos a seguir, muchas gracias por todo su amor y su apoyo, por depositar toda su confianza en mí, me ayudaron mucho, gracias.

A mi hermano por su cariño y apoyo incondicional.

A mis abuelas y abuelo, muchas gracias por estar atentos a mí, por su preocupación y todo su amor.

A mis tíos y mi primo que siempre supieron animarme en el momento justo.

A mi tía y mis primas que a pesar de estar lejos siempre se preocuparon por mí y me ayudaron muchísimo.

A mi novia, que fue otro de los puntos de apoyo que evitó que me derrumbara en muchas ocasiones, muchas gracias por todo tu amor.

A mi compañero de tesis y amigo Jorge Luis, por tu amistad, apoyo y ayuda, muchas gracias por ser mi hermano durante estos cinco años, sí se pudo.

A todos mis amigos y amigas del aula y la residencia que hicieron que mi vida en la universidad se sintiera como mi casa, con mi familia, muchas gracias por todos estos años de alegrías y preocupaciones, a todos los quiero muchísimo. A mi amigo Ihosbanny, que siempre estuvo en los momentos buenos y en los enredados, gracias hermano.

A mis tutoras, muchas gracias por toda su ayuda y paciencia, este logro es también de ustedes, de todo corazón muchísimas gracias.

A todos los profesores que me dieron el conocimiento y las herramientas para llegar hasta aquí, muchas gracias, nunca olvidaré sus enseñanzas.

Jorge Luis:

Inmensas gracias a mi tutora Adisley, por sus peleas, regaños y ayuda, además por la amistad que no solo se resume a este tiempo como su tesista.

A Indira, de corazón mil gracias por tanta paciencia y ayuda, por ese empeño y dedicación para con este duo, siempre voy a recordar lo importante que has sido.

A Glennis por convertirse sin que lo imagináramos en una tutora más, gracias por tu ayuda, tus consejos y tu esfuerzo para con nosotros, siempre te estaré agradecido.

A Breissy por ser mi amiga incondicional y por apoyarme en todo lo que he necesitado siempre.

A Ihosbanny, por cada una de esas veces que dijimos este año sí.

A Rafe y su esposa.

A mi familia del aula y apartamento que para mí nunca cambio de grupo solo se le sumaron más integrantes en aquel grupo dos, fue un placer compartir tan lindos años con ustedes, los quiere siempre su bro.

A Irina por haberme guiado y ayudado siempre.

A todos los profesores que contribuyeron para que llegara hasta aquí.

A todo mi equipo de futbol y al Grego.

A todas las personas que han contribuido en este tiempo con mi formación.

A mi familia que siempre me ha querido y apoyado, y a los amigos que también son mi familia.

A este duo de tesis que es una de las cosas más importantes que tengo en la vida, gracias mi hermano, no solo por esta tesis, gracias por todo lo que hemos pasado juntos desde que empezamos esta carrera, que para mí no marcará nunca el fin de esta amistad.

A lo más importante que tengo en la vida, mis padres: Mi padre por ser el ejemplo de incansable luchador, por su confianza en mí, por ser mi consejero y sobre todo mi amigo. Mi madre por ser siempre mi inspiración, ser mi guía, mi motivación, por ser tan dedicada a mí, por demostrarme tanto amor, por sufrir conmigo, por tantos momentos felices que me has dado, por confiar siempre en mí. Gracias por ser la razón más grande que tengo para luchar en la vida.

Dedicatoria

Luis Daniel:

A mi familia, en especial a mis padres y hermano está dedicado todo mi esfuerzo.

Jorge Luis:

Dedicado a mis padres por ser la motivación de cada cosa que persigo en mi vida, por ser la fuerza que me levanta en caso de fracaso, por ser mi orgullo y mi inspiración. A mi familia, amigos y a mis hermanas.

Resumen

En el proyecto Sistema Minero Cubano, se lleva a cabo el desarrollo de un sistema para el análisis y modelado de yacimientos minerales, Syam. Dicho sistema permite la interpretación geológica de los depósitos minerales, la modelación y estimación de los recursos minerales, así como el diseño, optimización y planificación de la actividad minera cubana. Cada nuevo yacimiento mineral analizado por Syam, parte de la importación de cuatros ficheros básicos cuyos datos pueden variar en correspondencia con los recogidos en el terreno por los especialistas mineros. Las tablas almacenadas en la base de datos correspondientes a cada fichero presentan datos comunes y otros que varían, lo cual provoca que las mismas crezcan en cantidad de columnas dificultando el manejo de estas.

Los desarrolladores acceden a los datos almacenados desde cualquier capa de las definidas por la arquitectura de la aplicación en que trabajen. Esto trae consigo el solapamiento de dichas capas de la aplicación y dificulta la migración entre sistemas gestores de bases de datos. El presente trabajo tiene como resultado un componente que permite la conversión a objetos a partir de la información almacenada en las tablas dinámicas manejadas por Syam en tiempo de ejecución, proceso que se conoce como mapeo objeto-relacional, lo que ayudará además, a corregir errores de arquitectura como solapamiento de capas y la no abstracción de la base de datos por parte de los desarrolladores del sistema.

El desarrollo del componente y la documentación generada están guiados por la metodología AUP. Como herramientas y tecnologías utilizadas para el desarrollo de la aplicación se utilizaron: UML v2.1 como lenguaje de modelado y Visual Paradigm v8.0 como herramienta Case para garantizar el modelado de los artefactos del diseño. Como lenguaje de programación para la implementación del componente se utilizó C++ con QT como marco de trabajo y QTCreator v5.2 como IDE de desarrollo, además de libpq v5 como biblioteca para realizar consultas al servidor PostgreSQLv8.0. Como sistemas gestores de bases de datos se utilizó PostgreSQL y SQLite. Como herramientas para la administración de las base de datos se utilizaron PgAdmin v9.4 y SQLite Manager v1.2.

Palabras clave: componente, estructura variada, mapeo objeto-relacional.

Abstract

In the Cuban Mining System project it is carried out to develop a system for analysis and modeling of mineral deposits, Syam. This system allows the geological interpretation of mineral deposits, modeling and estimation of mineral resources, as well as the design, planning and optimization of the Cuban mining. Each mineral deposit analyzed by Syam, begins with the importation of the four basic files whose data may vary in correspondence with those collected by the miners specialists on the field. The tables stored in the database corresponding to each file have common data and other that vary, which causes them to grow in columns number making harder the handling of these.

System developers currently have to access to the database that stores the information gathered from the imported files from any architecture's define layers of the system. This brings with it the overlapping of the application layers and makes harder the migration between databases management systems. This work has resulted in a component that that allow the conversion to objects from the stored information on the dynamics tables handled by Syam in runtime, process known as object-relational mapping, which will also help to correct the architecture problems as overlapping layers and the not abstraction of the database by the project developers.

The development of the component and the documentation that it generates are guided by the development methodology AUP. As tools and technologies used for application development: UML v2.1 as modeling language, Visual Paradigm v8.0 as a Case tool ensuring the design artifacts modeling. As a programming language for the implementation of component it was used C ++ with the framework QT and QT Creator v5.2 as development IDE, as well libpq v5 as library for query to the server PostgreSQL v8.0. As database management systems was used PostgreSQL and SQLite. As tools for managing the database were used PgAdmin III v9.4 and SQLite Manager v1.2.

Keywords: *component, dynamic structure, object-relational mapping.*

Índice de Contenido

Índice de Figuras

Índice de Tablas

Introducción

El gran cúmulo de información que se genera en las empresas ha obligado a las mismas a optar por la utilización de las Tecnologías de la Información y las Comunicaciones (TICS) para un mejor manejo de esta gran cantidad de datos. De esta forma se logra brindar un mejor servicio, reducir costos y aumentar la eficiencia de dichas instituciones.

Las empresas mineras a nivel nacional han decidido avanzar en el aspecto de informatización de sus sistemas, para aumentar la calidad de sus procesos. Presentan interés en la gestión e integración de los datos generados por las perforaciones realizadas, para transformarlos en información para sus negocios mediante el uso de sistemas informáticos.

Con lo planteado anteriormente, dichas empresas deben adquirir software privativo a altos costos por concepto de licencia, actualizaciones, soporte y capacitación. Apareciendo como inconveniente que la mayoría del software disponible en esta rama es propiedad o ha sido adquirido por corporaciones norteamericanas lo que trae consigo que las pequeñas empresas mineras no cuenten con el presupuesto necesario para costear un software de este tipo.

La Universidad de las Ciencias Informáticas (UCI) se encuentra inmersa en la automatización de las empresas del país. En el departamento de Geoinformática, perteneciente al centro de desarrollo de Geoinformática y Señales Digitales (GEySED) de la UCI, se trabaja en el proyecto Sistema Minero Cubano, en el que se lleva a cabo, en colaboración con entidades geólogo-mineras del país, el desarrollo de un sistema para el análisis y modelado de yacimientos minerales, denominado Syam. Este sistema permite la interpretación geológica de los depósitos minerales, la modelación y estimación de los recursos minerales, así como el diseño, optimización y planificación de la actividad minera cubana. Dicho sistema está compuesto por los módulos: Bases de datos, Visualización y Núcleo del sistema; en los cuales se gestionan las diferentes funcionalidades del mismo.

Cada nuevo proyecto creado con el sistema Syam, parte de la importación de cuatros ficheros fundamentales, estos son: pozo, inclinometría, muestras y litología; que pueden tener más o menos datos en dependencia de los recopilados en el terreno por los especialistas geólogos o mineros, los cuales guardan toda la información referente a los pozos de perforación en un yacimiento mineral.

El fichero pozo guarda las coordenadas (x, y, z) de la ubicación exacta de cada pozo en el terreno. El fichero inclinometría almacena la información del ángulo de perforación de cada pozo. El fichero muestras documenta la concentración de minerales por muestras encontradas, así como sus intervalos de aparición relacionados con el pozo al que pertenecen. Y el fichero litología almacena las diferentes capas litológicas. Estos ficheros se encuentran relacionados entre sí por el fichero pozo, donde toda la información almacenada en el resto de los ficheros pertenece a un pozo específico y se vincula con el mismo a través de un identificador de pozo. Esto significa que para analizar un pozo en su totalidad se hace necesario la utilización de la información almacenada en los cuatro ficheros mencionados.

La información fundamental con la que trabaja el sistema se almacena en estos ficheros, por lo que los datos que contienen, se manejarán en clases que reflejan su comportamiento en la base de datos. Las características de los objetos creados a partir de estas clases, conocidas como atributos, coinciden con las columnas de sus tablas correspondientes en la base de datos, y se tratan como objetos de negocio.

Estos ficheros contienen como datos comunes para cada proyecto, las coordenadas que marcan la ubicación del pozo, los ángulos de inclinación asociados al pozo, la concentración de minerales en cada muestra, los intervalos en los que aparecen dichas muestras, así como las capas litológicas que existen en cada perforación. Además de estos datos los ficheros pueden contener otros datos en correspondencia con la información recogida en el terreno por los especialistas. Dicha información es guardada en la base de datos en forma de nuevas columnas, lo que trae consigo que la cantidad de columnas de las tablas pueda aumentar o no dependiendo de la información contenida en los ficheros.

Por otra parte el manejo de una estructura dinámica complejiza el trabajo de los programadores, ya que los obliga a conocer la estructura creada por la aplicación para administrar los datos de cada proyecto, teniendo que dominar el lenguaje SQL para, en cualquier capa de la aplicación, escribir código de este tipo. Lo anteriormente planteado trae como consecuencia que si se cambia el sistema gestor de base de datos, se tengan que modificar todos los ficheros en que se tiene este tipo de código, generando así que el manejo de los datos sea lento y poco efectivo.

A partir de la problemática antes planteada surge el siguiente **problema de investigación**: ¿Cómo administrar en la base de datos del sistema Syam las variaciones en tiempo de ejecución de las columnas de las tablas?

Para darle solución al problema planteado se define como **objetivo general**: Desarrollar un componente para el mapeo objeto-relacional que permita la administración en la base de datos del sistema Syam de las variaciones en tiempo de ejecución de los atributos de los objetos de negocio.

En el presente trabajo se define como **objeto de estudio**: El proceso de mapeo objeto-relacional. Enmarcado en el **campo de acción:** Proceso de mapeo objeto-relacional para tablas dinámicas en Syam.

Para guiar la investigación se definen las siguientes **preguntas de investigación:**

1. ¿Cuáles son las bases teóricas que fundamentan el desarrollo del componente para la administración de las variaciones en tiempo de ejecución de los atributos de los objetos del negocio en la base de datos del sistema Syam?

2. ¿Cuáles son las características y capacidades que debe tener el componente?

3. ¿Cómo proporcionar una imagen completa de los dominios funcionales, de comportamiento y de datos desde una perspectiva de implementación del componente?
4. ¿Cómo validar el correcto funcionamiento del componente?

Para dar cumplimento al objetivo general se definen como **tareas de la investigación**:

1. Análisis de los elementos conceptuales implicados en el desarrollo del componente para la administración de las variaciones en tiempo de ejecución de los atributos de los objetos del negocio en la base de datos del sistema Syam.

2. Selección de la metodología y herramientas a utilizar para guiar el desarrollo del componente.

3. Identificación de las necesidades funcionales y tecnológicas del componente para su correcto funcionamiento.

4. Identificación de los principios arquitectónicos del componente para definir su estructura global.

5. Identificación de los principios de diseño y funcionamiento del componente para guiar el proceso de implementación.

6. Implementación de los principios de diseño y funcionamiento para satisfacer las necesidades funcionales y tecnológicas del componente.

7. Validar el componente mediante pruebas de software para comprobar su correcto funcionamiento.

Métodos Teóricos

Analítico-Sintético: Se utiliza este método para analizar los conceptos teóricos existentes sobre el mapeo objeto relacional, en lo adelante ORM, y extraer las características que debe cumplir el Componente de mapeo objeto-relacional para el sistema Syam.

Modelación: Se utiliza para a través de los modelos de análisis, diseño e implementación representar la realidad objetiva en que se enmarca el sistema y para modelar la solución del *software*.

Métodos Empíricos

Análisis Estático: Para realizar un estudio en profundidad de las soluciones existentes que realizan mapeo objeto-relacional con el fin de identificar elementos claves que contribuyen a la solución del problema planteado.

Técnicas de recopilación de información

Tormenta de ideas: Se emplea con el objetivo de identificar las características y capacidades con las que debe cumplir el Componente de mapeo objeto-relacional para el sistema Syam. Teniendo en cuenta los criterios del jefe de proyecto, los tutores de la investigación y los autores.

Resultados esperados

1. Componente de mapeo objeto-relacional para el sistema Syam.

2. Documentación técnica generada en el proceso de desarrollo correspondiente al componente.

Estructuración capitular

El trabajo de diploma se divide en tres capítulos, de los cuales su estructura es la siguiente:

Capítulo 1. Fundamentación teórica sobre el proceso de mapeo objeto-relacional: En este capítulo se definen los principales conceptos a tratar relacionados con el mapeo objeto-relacional, además de realizarse una profundización del objeto de estudio. Se lleva a cabo un estado del arte sobre ORM existentes a nivel internacional. Además de definir la metodología, las herramientas y tecnologías a utilizar en el desarrollo del componente para el mapeo objeto-relacional para el sistema Syam.

Capítulo 2. Análisis y diseño del componente para el mapeo objeto-relacional del sistema Syam: En este capítulo se indican los procesos actuales mediante el modelo de dominio, se identifican los requisitos funcionales y no funcionales, además del diagrama de casos de uso del sistema donde se encapsulan los requisitos. Es definida la arquitectura del sistema, además de los patrones arquitectónicos y de diseño por los cuales se guiará el desarrollo del componente. En este capítulo se realiza además, el modelo de diseño del componente, para organizar y preparar el terreno para una implementación con mayor eficiencia, que contribuye a un producto final con calidad.

Capítulo 3. Implementación y Pruebas del Componente para el mapeo objeto-relacional en el sistema Syam: En este capítulo se realizan tareas definidas por la metodología AUP, se realiza el modelo de implementación para el desarrollo del componente, en el cual están contenidos los diagramas de componentes y de despliegue del sistema. Quedan definidos cuales son los estándares de codificación por los cuales se rige la implementación del componente. Así como las pruebas a realizarse una vez culminado el desarrollo del software, además de la documentación del resultado de las mismas.

Capítulo 1: Fundamentación teórica sobre el proceso de mapeo objeto-relacional

1.1 Introducción

En el presente capítulo se exponen las definiciones que permitirán esclarecer la problemática en cuestión. Se realiza un análisis de soluciones existentes, las cuales sirven de apoyo a la investigación y la obtención de sus resultados. Además se estudian las tecnologías y herramientas a utilizar en el desarrollo y documentación de la presente investigación.

1.2 Análisis de los conceptos asociados al dominio del problema

La gran cantidad de información relacionada con las perforaciones realizadas para la búsqueda de yacimientos minerales, surge de los datos recogidos en el terreno por los especialistas geólogos mineros, dichos datos son almacenados en una base de datos relacional. Para un mejor entendimiento de la forma en que son almacenados los datos, se hace necesario definir los elementos relacionados con la temática en cuestión.

Partiendo desde la definición más sencilla, para entender la relación existente entre todos los componentes que permiten el almacenamiento y manejo de la información, se hace referencia a los datos.

Dato.

Según la Real Academia Española ente sus definiciones se encuentra que dato es: "*Información dispuesta de manera adecuada para su tratamiento por un ordenador."*

Pero este significado, no es están abarcador como la definición a utilizar en adelante, que se refiere al dato como: "*el elemento de información recogido durante la investigación para llegar al conocimiento exacto de lo que se busca. La recolección de los datos está entre las tareas más difíciles e importantes de cualquier investigación."* (González Castellanos, y otros, 2003)

Los datos obtenidos deben ser almacenados para que perduren en el tiempo, y puedan ser utilizados y analizados por los especialistas. Lo cual se logra a travéz del uso de una Base de Dato (BD). Algunas definiciones de BD se muestran a continuación:

Una definición bien explicita es la mostrada por Eva Gómez Ballester y otros autores en el libro Base de datos 1: *"Una base de datos es una serie de datos relacionados que forman una estructura lógica, es decir una estructura reconocible desde un programa informático. Esa estructura no sólo contiene los datos en sí, sino la forma en la que se relacionan."* (Gómez Ballester, y otros, 2007)

También Rafael Camps Paré y otros autores resumieron que: *"Una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada, a pesar de que debe permitir utilizaciones varias y simultáneas."* (Camps Paré, y otros, 2005)

Otra visión muy similar de mismo término es la expuesta por la Dra. Rosa María Matos: "*Conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora. O sea, que una BD puede considerarse una colección de datos variables en el tiempo."* (Matos, 1999)

Y la definición mostrada en el libro Introducción a los Sistemas de Bases de Datos por C.J. Date en 2001: "*Un sistema de bases de datos es básicamente un sistema computarizado para llevar registros. Es posible considerar a la propia base de datos como una especie de armario electrónico para archivar; es decir, es un depósito o contenedor de una colección de archivos de datos computarizados. Los usuarios del sistema pueden realizar una variedad de operaciones sobre dichos archivos."* (Date, 2001). Esta será la definición de Bases de Datos a utilizar en el transcurso de la investigación.

Para realizar el manejo de los datos y facilitar la inserción, actualización y/o eliminación se hace necesario la utilización de Sistemas Gestores de Base de datos (SGBD). Aunque existen diferentes SGBD, básicamente todos tienen un mismo fin. Algunos autores lo definen de diferentes puntos de vista, pero todos convergen en un mismo significado:

Tal es el caso de la brindada por la Dra. Rosa María Matos: "*El software que permite la utilización y/o la actualización de los datos almacenados en una (o varias) base(s) de datos por uno o varios usuarios desde diferentes puntos de vista y a la vez, se denomina SGBD."* (Matos, 1999)

Eva Gómez Ballester y otros autores en el libro Base de datos 1, lo define de la forma siguiente: *"Un SGBD es un programa de ordenador que facilita una serie de herramientas para manejar bases de datos y obtener resultados (información) de ellas. Además de almacenar la información, se le pueden hacer preguntas sobre esos datos, obtener listados impresos, generar pequeños programas de mantenimiento de la BD, o ser*

utilizado como servidor de datos para programas más complejos realizados en cualquier lenguaje de programación." (Gómez Ballester, y otros, 2007)

También queda esclarecido el término en la definición presentada por los autores del libro Fundamentos de Bases de datos, que se refiere a: *"El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto práctica como eficiente…Uno de los propósitos principales de un sistema de bases de datos es proporcionar a los usuarios una visión abstracta de los datos. Es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos."* (Silberschatz, y otros, 2002). Con esta clara definición es la que estará utilizando en todo el trayecto de nuestra investigación.

Las bases de datos relacionales solo soportan tipos de datos simples, lo que dificulta el manejo y uso de su contenido desde un entorno de Programación Orientada a Objeto (POO): que no es más que un paradigma de programación que utiliza objetos en sus interacciones, incluyendo para el diseño de programas informáticos técnicas como la herencia, cohesión, polimorfismo, abstracción, encapsulamiento entre otras. Por lo que se hace necesario la conversión de estos tipos de datos simples en objetos, de la misma manera que los objetos deberán ser insertados en la base de datos como datos simples.

Aunque para muchos programadores en algunos casos, es innecesario un **mapeo objeto-relacional,** es necesario conocer qué significa. Lograr entender su significado comienza por atender a las definiciones dadas por varios especialistas en el tema:

En el artículo Acceso a datos en aplicaciones multihilos orientadas a objetos los autores Yadira Lizama Mue y Lissuan Fadragas Artiles publicado en 2010 se resume con claridad que es un ORM: *"Los ORM permiten la coexistencia del modelo relacional de la base de datos y el diseño orientado a objetos de las aplicaciones en nuestros entornos de software y la comunicación entre ellos, garantizando cierta abstracción del programador sobre el modelo relacional pudiendo manipular la persistencia de sus objetos con una mínima interacción con el mismo. En su mayoría están compuestos por librerías, clases y formas descriptivas que permiten el mapeo de las clases. Soportan al menos el mapeo a un tipo de bases de datos, son utilizados únicamente por las aplicaciones implementadas con el mismo lenguaje de programación y, por supuesto, mapean bases de datos relacionales u objeto-relacionales únicamente."* (Lizama Mué, y otros, 2010)

También en 2010 Lisandra Díaz Romero y otros autores definen que ORM: *"Es una técnica de programación para manejar datos entre sistemas que utilizan programación orientado a objetos y bases de datos relacionales. Básicamente es manejar una base de datos relacional a través de objetos nativos de un lenguaje específico."* (Díaz Romero, y otros, 2010)

El termino es definido por Jonatan Alejandro Rico y Zapopan Irvine en 2013 como: *"El Mapeo Objeto-Relacional (Object-Relational Mapping, en inglés) es una capa de persistencia que conecta Objetos en un sistema Orientado a Objetos con información almacenada en una base de datos Relacional. El uso del Mapeo Objeto-Relacional permite aplicar el diseño, análisis y técnicas de programación Orientada a Objetos mientras se mantienen ocultos los detalles específicos de la Base de datos Relacional."* (Rico, y otros, 2013)

Por tanto será tomado como referente la definición de mapeo objeto-relacional que expresan anteriormente Jonatan Alejandro Rico y Zapopan Irvine.

1.3 Análisis de ORM existentes

Las herramientas ORM contribuyen a reducir significativamente la implementación relacionada con el código que permite el acceso a los datos, además admiten diseñar un modelo entidad relación y generar a partir de ello un esquema de base de datos real al mismo tiempo que se establece el mapeo de objetos/entidades del dominio y la base de datos. En caso de existencia de la base de datos, generalmente los ORM son capaces de generar el modelo de datos (este modelo considera la base de datos como una colección de relaciones) y mapear los objetos de igual forma.

En la actualidad, existe gran cantidad de ORM implementados en diferentes lenguajes de programación. Los cuales son muy similares en su forma de interactuar con las BD relacionales, y muchas de sus funcionalidades son semejantes. Como parte del estudio de los ORM implementados y utilizados en la actualidad, podemos hacer referencia a Hibernate, que es clasificado como un ORM de software libre.

Hibernate ORM: (Hibernate, 2015): Es una herramienta de Mapeo Objeto Relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los *beans* (usan para encapsular varios objetos en un único objeto) de las entidades que permiten establecer estas relaciones. Es software libre, distribuido bajo los términos de la licencia GNU LGPL.

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizando, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

Ofrece también un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), al mismo tiempo que una API para construir las consultas (conocida como "criteria").

Dentro de sus principales características se encuentran las siguientes, según:

- Se clasifica entre los ORM de software libre por contar con una licencia LGPL (Lesser GNU Public License). La cual permite modificación al obtener su código fuente para la preparación de nuevas tareas o productos.
- Entorno de trabajo: Su plataforma permite su utilización en redes de computadores que estén realizando procesos en paralelos así su arquitectura de caché de doble capa sería más eficaz.
- Funcionalidad: Implementa la función de metadatos las cuales son palabras claves que agrupan grandes volúmenes de información.
- Sistema de archivos: Su formato de archivos en el momento de los mapeos es XML ofreciendo ventajas entre esas la representación de la información y el permitir ejecutar modificaciones.

Como ventaja que brinda Hibernate se tiene que:

- Permiten la integración de cualquier sistema de acceso a base de datos relacionales. Siendo de esta manera más productiva y su tiempo de desarrollo es más corto.

Otro ORM muy reconocido y utilizado a nivel internacional es Doctrine ORM el cual se muestra a continuación:

Doctrine ORM: (Doctrine, 2014) El Proyecto Doctrine contiene varias bibliotecas PHP que se centran principalmente en el almacenamiento de base de datos y el mapeo a objeto. Presenta integración con muchos frameworks (Symfony, Zend Framework, CodeIgniter, FLOW3, Lithium). Los proyectos básicos son un ORM y la capa de abstracción de base de datos, por sus siglas en inglés (DBAL) en la que se basa. Doctrine se ha beneficiado enormemente de conceptos del ORM Hibernate y los ha adaptado para encajar al lenguaje PHP.

Doctrine permite la persistencia de objetos completos a la base de datos y recuperar objetos completos desde la base de datos. Esto funciona asociando una clase PHP a una tabla de la base de datos, y las propiedades de esa clase PHP a las columnas de la tabla. Entre los métodos que implementa Doctrine se encuentran algunos como:

- Método *persist*, utilizado a través del *EntityManager* (clase controladora de Doctrine). El método persist se utiliza para notificar al *EntityManager* que se realizará, por ejemplo, una inserción en la base de datos.
- Método *flush,* este método se utiliza específicamente para comenzar la transacción que almacenará en la base de datos los cambios realizados.

La utilización de estos métodos permite agregar todas las transacciones realizadas (INSERT, UPDATE, DELETE) dentro de una sola, la cual se ejecuta al utilizar el método *flush*. Utilizando esta vía el rendimiento de escritura en la base de datos es significativamente mayor a si cada transacción se realizara una a una para cada entidad.

Algunas de las principales ventajas que brinda Doctrine:

- Extremadamente flexible, poderoso mapeo de objeto.
- Soporte tanto para programación de base de datos de alto nivel y la de bajo nivel. (Doctrine, 2014)

Estos ORM no conforman una solución factible pues no se encuentran implementados en el leguaje de desarrollo C++.

QxORM

Es una biblioteca C++ diseñada para proporcionar mapeo de objeto-relacional (ORM) característico para los usuarios de C++. Ofrece varias funcionalidades como: *Persistencia:* La comunicación con una gran cantidad de bases de datos (con relaciones de uno a uno (1-1), uno a muchos (1-N), muchos a uno (N-1) y las relaciones de muchos a muchos (N-N)). *Serialización:* Todas las clases definidas por QxOrm puede serializar en formato binario y xml*. Reflexión o Introspección:* El acceso a las definiciones de las clases, recuperar propiedades y llamar a los métodos de las clases. Soporta las características de la programación orientada a objetos, entre las que se encuentra la herencia y el polimorfismo. (QxORM, 2011)

ODB

Se encuentra doblemente licenciado, bajo los términos de GNU General Public License y también bajo licencia privativa. A diferencia de otros ORM para C++, permite la generación automática de código para el mapeo de objetos, además de generar el esquema de base de datos a partir de las declaraciones de las clases, para lo cual se deben colocar un conjunto de directivas embebidas en el código. Estas directivas permiten controlar diferentes aspectos, tales como los nombres de las tablas y las columnas, el mapeo de tipos de datos C++ a tipos SQL, entre otros. Para realizar esta última tarea se apoya en el compilador ODB, que solo genera códigos C++ que a su vez pueden ser compilados por otros compiladores. Proporciona una interfaz orientada a objetos que permite realizar diferentes operaciones sobre los objetos persistentes, así como el lenguaje ODB Query Language para la escritura de consultas integradas al código C++. (CC, 2009-2014)

A continuación se evidencian las características principales que presentan QxORM y ODB realizándose una comparación teniendo en cuenta SGBD, licencia y plataformas en las que pueden ser utilizadas.

Tabla 1.Principales características de QxORM y ODB

	SGBD	Licencia	Código	Plataformas
QxORM	PostgreSQL, IBM DB2, MySQL, SQL Server, Oracle, SQLite, Borland InterBase y Sybase Adaptive Server.	GNU General Public License.	Código abierto.	Windows, Mac OS X y Linux.
ODB	PostgreSQL, SQLite, MySQL, SQL Server y Oracle.	Licencia propietaria y GNU General Public License.	Código abierto y comercial.	Windows, Mac OS X y Linux.

Como QxORM y ODB han sido implementados en el lenguaje C++, el cual es un lenguaje compilado, los objetos generados a partir de las bases de datos tienen sus atributos y métodos predefinidos, lo que impide adicionar atributos a las tablas en tiempo de ejecución. Por lo que se hace imposible para estos ORM el manejo de nuevos atributos y variaciones del número de columnas en las tablas de forma dinámica.

Debido a lo analizado, se concluye que los ORM valorados no son una solución para las necesidades de Syam. Porque algunos de ellos son software privativo y su costo no está al alcance del proyecto. Además de que ninguna de las soluciones existentes cubre la necesidad del manejo de una estructura que permita las variaciones en tiempo de ejecución de los atributos de los objetos del negocio.

Aunque no son una solución, pueden ser tomadas algunas características de estos ORM, para la implementación de un componente ORM ajustado a las características específicas del sistema.

Algunas de las ventajas que contiene ODB y pueden ser útiles para la implementación del Componente ORM, para garantizar la persistencia de los objetos son:

- Código conciso: Con ODB se pueden ocultar los detalles de la base de datos correspondiente, la lógica de la aplicación está escrita utilizando el vocabulario natural por lo que es más simple y fácil de leer y entender.
- El rendimiento óptimo: ODB ha sido diseñado para un alto rendimiento y bajo costo operativo de memoria.
- Mantenibilidad: Generación automática de código y la evolución del esquema de base de datos minimizan el esfuerzo necesario para adaptar la aplicación a los cambios en las clases persistentes. El código de conversión de base de datos se mantiene separado de las declaraciones de clases y la lógica de la aplicación. Brindando con esto facilidad de mantención y depuración a la aplicación.

Por otra parte QxORM está conformado por varios módulos, que garantizan algunas funcionalidades que bien pudieran ser de ayuda para el componente ORM para el sistema Syam:

- Persistencia: La comunicación con una gran cantidad de bases de datos (con 1-1, 1-n, n-1 y las relaciones n-n).
- Reflexión (o introspección): Acceso a definiciones de las clases, recuperar propiedades y llamar a los métodos de las clases.
- Generar secuencia de comandos DDL (lenguaje de definición de datos) SQL de forma automática (esquema de base de datos).

1.4 Metodología de desarrollo de software

Para el diseño y desarrollo de proyectos de software se aplican metodologías, modelos y técnicas que permiten resolver los problemas que se presenten durante el desarrollo de los mismos. Una metodología

impone un proceso de forma disciplinada sobre el desarrollo de software con el objetivo de hacerlo más predecible y eficiente. Las metodologías definen una representación que permite facilitar la manipulación de modelos, y la comunicación e intercambio de información entre todas las partes involucradas en la construcción de un sistema. (Sommerville, 2005) La metodología de desarrollo de software es un enfoque estructurado para el desarrollo de software que incluye modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos. Existen dos grandes grupos de metodologías: las robustas o tradicionales y las ágiles o ligeras.

1.4.1 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. Dentro de estas se destacan las metodologías de desarrollo de software SCRUM, XP y AUP las cuales se analizan a continuación.

SCRUM: Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto. Éstas son las verdaderas protagonistas, especialmente la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración. (Schwaber, y otros, 2001)

XP: Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent

Beck, el padre de XP, describe la filosofía de XP, sin cubrir los detalles técnicos y de implantación de las prácticas. (Beck, 1999)

Proceso Ágil Unificado (AUP): El proceso unificado ágil es un enfoque híbrido de modelado creado por Scott Ambler cuando combinó el Proceso de Desarrollo Unificado (RUP) a los métodos ágiles (MA), establece un Modelo más simple que el que aparece en RUP por lo que reúne en una única disciplina, Modelo, las disciplinas de Modelado de Negocio, Requisitos y Análisis y Diseño. El resto de disciplinas (Implementación, Pruebas, Despliegue, Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP. (Ambler, 2005)

Estas disciplinas mencionadas son:

1. Modelo: El objetivo de esta disciplina es entender el negocio de la organización, el problema de dominio que se abordan en el proyecto, y determinar una solución viable para resolver el problema de dominio.

2. Aplicación: El objetivo de esta disciplina es transformar su modelo (s) en código ejecutable y realizar un nivel básico de las pruebas, en particular, la unidad de pruebas.

3. Prueba: El objetivo de esta disciplina consiste en realizar una evaluación objetiva para garantizar la calidad. Esto incluye la búsqueda de defectos, validar que el sistema funciona tal como está establecido, y verificando que se cumplan los requisitos.

4. Despliegue: El objetivo de esta disciplina es la prestación y ejecución del sistema y que el mismo este a disposición de los usuarios finales.

5. Gestión de configuración: El objetivo de esta disciplina es la gestión de acceso a herramientas de su proyecto. Esto incluye no sólo el seguimiento de las versiones con el tiempo, sino también el control y gestión del cambio para ellos.

6. Gestión de proyectos: El objetivo de esta disciplina es dirigir las actividades que se lleva a cabo en el proyecto. Esto incluye la gestión de riesgos, la dirección de personas (la asignación de tareas y el seguimiento de los progresos), coordinación con el personal y los sistemas fuera del alcance del proyecto para asegurarse de que es entregado a tiempo y dentro del presupuesto.

7. Entorno: El objetivo de esta disciplina es apoyar el resto de los esfuerzos por garantizar que el proceso sea el adecuado, la orientación (normas y directrices), y herramientas (hardware y software) estén disponibles para el equipo según sea necesario.

AgileUP recomienda incluir prácticas que tienen como fin primordial la agilización de las actividades de implementación: programación por pares, desarrollo guiado por pruebas (TDD), integración continua, modelar antes de codificar, adoptar un estándar de codificación, refactorizar el código y los esquemas de bases de datos, y tener ambientes separados para el desarrollo, las pruebas y la producción. El Proceso de Integración se traduce en la actividad continua de compilación del sistema. Esta compilación se realiza cada vez que el código fuente cambia. Así, el repositorio de código fuente se encuentra, en todo momento, en un estado estable respecto a las nuevas funcionalidades que han sido agregadas al producto.

El Proceso Unificado Ágil consta de cuatro fases que el proyecto atraviesa de forma secuencial. Dichas fases son, al igual que en el Proceso de Desarrollo Unificado:

1. **Iniciación**. El objetivo de esta fase es identificar el alcance inicial del proyecto, una arquitectura potencial para el sistema y obtener, si procede, financiación para el proyecto y la aceptación por parte de los promotores del sistema.

2. **Elaboración.** Mediante esta fase se pretende identificar y validar la arquitectura del sistema.

3. **Construcción.** El objetivo de esta fase consiste en construir software desde un punto de vista incremental basado en las prioridades de los participantes.

4. **Transición.** En esta fase se valida y despliega el sistema en el entorno de producción.

La metodología AUP es aplicable a proyectos cortos y de pocos integrantes. AUP maneja los conceptos de entregables, productos de trabajo empresariales y otros productos de trabajo. Los entregables son aquellos productos de trabajo que deben ser producidos. Los otros productos de trabajo son aquellos que no se requiere mantenerlos en el tiempo. Los productos de trabajo empresariales son aquellos que son mantenidos dentro de la organización y son compartidos entre proyectos. Entre las prácticas recomendadas están: mantener los productos de trabajos simples y concisos; menos documentación que en RUP; trabaja de manera cercana al cliente/usuario y se produce sólo aquello que realmente se necesita. Por estas razones se decidió usar dicha metodología para el desarrollo del componente. (Rodríguez, 2014)

1.5 Herramientas y tecnologías

Para el desarrollo del Componente de mapeo objeto-relacional para el sistema Syam, se hace necesario la utilización de herramientas y tecnologías, que permitan el análisis, diseño, modelado y desarrollo del componente.

1.5.1 Lenguaje de modelado

UML v2.1 (Lenguaje Unificado de Modelado) como su nombre indica es un lenguaje de modelado para describir métodos o procesos. Con el objetivo de detallar los artefactos del sistema, construir y describir un modelo en sí. Incluye conceptos como funciones del sistema, procesos de negocio además de aspectos concretos como esquemas de bases de datos y lenguajes de programación. En esencia permite visualizar, documentar, especificar y construir software orientado a objetos. (IBM Software Group, 2005)

1.5.2 Herramienta CASE

Para garantizar el modelado de los artefactos del diseño, con el fin de facilitar la comprensión y guiar el desarrollo del componente, es preciso valorar que herramienta CASE (Computer Aided Software Engineering, “siglas del inglés” Ingeniería de Software Asistida por Ordenador) será recomendable utilizar.

Visual Paradigm v8.0: Es una herramienta CASE, que utiliza el lenguaje UML profesional, que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Su diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad. Uso de un lenguaje estándar común a todo el equipo de desarrollo para facilitar la comunicación. (Visual Paradigm, 2015)

1.5.3 Lenguaje de Programación

El lenguaje a utilizar para implementar las funcionalidades del componente mapeo objeto-relacional, es el lenguaje C++, ya que este también es el lenguaje con el cual se desarrolla el sistema al que será integrado el componente mencionado.

C++ es un lenguaje imperativo orientado a objetos derivado del C. En realidad un superconjunto de **C**, que nació para añadirle cualidades y características de las que carecía. El resultado es que como su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a

bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. Respecto a su antecesor, se ha procurado mantener una exquisita compatibilidad hacia atrás por dos razones: poder reutilizar la enorme cantidad de código **C** existente, y facilitar una transición lo más fluida posible a los programadores de **C** clásico, de forma que pudieran pasar sus programas a **C++** e ir modificándolos (haciéndolos más "**++**") de forma gradual.

En el diseño del C++ prima sobre todo la velocidad de ejecución del código, que incluye a diferencia del lenguaje C excepciones, sobrecarga de operadores, *templates* o plantillas.

1.5.4 Entorno de desarrollo

Qt es una aplicación multi-plataforma y un *framework* de interfaz de usuario utilizado para el desarrollo de aplicaciones con soporte a más de una docena de plataformas. El *framework* de Qt se compone de módulos multi-plataformas de clases C ++, bibliotecas Qt y un entorno de desarrollo integrado con Qt Creator IDE y varias herramientas. (The Qt Company, 2013)

QtCreator v5.2 es un IDE (Entorno de desarrollo integrado) para el desarrollo de aplicaciones. Es soportado por sistemas operativos como GNU/Linux, Mac OS X, Windows XP, Vista, Windows 7, por lo que es multiplataforma y además software libre. Permite construir interfaces de usuario complejas de una forma visual y rápida, ya que incluye un editor de texto con autocompletado, diseñador de interfaces gráficas, gestión de proyectos, sistema de depuración e integración con sistemas de control de versiones. (The Qt Company, 2013)

LibPQ v5: Es la interfaz de la aplicación de C del programador para PostgreSQL. Es un conjunto de funciones que permiten a los programas cliente pasar consultas al servidor PostgreSQL y recibir los resultados de estas consultas. Libpq es también el motor subyacente de varias otras interfaces de aplicaciones de PostgreSQL, incluyendo las escritas para C ++, Perl, Python, Tcl y ECPG . 1.5.5 Sistemas Gestores de Bases de Datos. (PostgreSQL , 2015)

PostgreSQL v9.4: Es sistema gestor de bases de datos de código abierto, diseñado para administrar grandes cantidades de datos. Puede ser utilizado en sistemas operativos como: Windows, Mac OS, Unix, Linux, entre otros. Contiene las características de una base de datos profesional como lo son: tipos de datos definidos por usuarios, triggers, funciones, secuencias, procedimientos almacenados, reglas, relaciones, vistas. Incluye la mayoría de tipos de datos, además de soporta lenguajes de programación como: Java,

PHP, Python, C, C++, Perl, entre otros. PostgreSQL se ha preocupado por ser una solución real a los complejos problemas del mundo empresarial y a la vez mantener la eficiencia al consultar los datos. Con ese fin, se han desarrollado y añadido al PostgreSQL las más interesantes y útiles características que antes sólo podían hallarse en sistemas manejadores de bases de datos comerciales como Oracle, DB2 o Sybase. (PostgreSQL , 2015)

SQLite III es una herramienta de *software* libre, que permite almacenar información en dispositivos fijos de una forma eficaz, potente, rápida y en equipos con pocas capacidades de hardware. SQLite implementa el estándar para la sintaxis y semántica de los lenguajes de bases de datos SQL92 y también agrega extensiones que facilitan su uso en cualquier ambiente de desarrollo. Esto permite que SQLite soporte desde las consultas más básicas hasta las más complejas del lenguaje SQL, y lo más importante es que se puede usar tanto en dispositivos móviles como en sistemas de escritorio, sin necesidad de realizar procesos complejos de importación y exportación de datos, ya que existe compatibilidad al 100% entre las diversas plataformas disponibles, haciendo que la portabilidad entre dispositivos y plataformas sea transparente**.** SQLite está construida en C, lo cual facilita la migración a diversas plataformas de sistemas operativos y de dispositivos. Dado que una base de datos de SQLite se almacena por completo en un solo archivo, está puede ser exportada a cualquier otra plataforma y tener interoperabilidad al 100% sin ningún requerimiento de programación adicional o cambios de configuración. (SQLite, 2014)

1.5.6 Herramienta de administración de Bases de Datos

PgAdmin III v9.3 es una herramienta con interfaz visual para gestionar los datos de un servidor de bases de datos con PostgreSQL, además de permitir la creación de consultas y vistas, permite gestionar la seguridad de acceso a la base de datos. La conexión al servidor puede hacerse a través de TCP/IP o *sockets* de dominio Unix, y puede ser encriptado SSL para la seguridad. PgAdmin es desarrollado por una comunidad de expertos de PostgreSQL en todo el mundo y está disponible en más de una docena de idiomas. Es un software libre publicado bajo la licencia BSD. (PgAdmin , 2015)

SQLite Manager v1.2: Es un sistema de gestión de bases de datos de gran alcance para las bases de datos SQLite, con una gran velocidad y características avanzadas. SQLiteManager permite trabajar con una amplia gama de bases de datos SQLite 3. Puede realizar operaciones básicas como crear y navegar por tablas, vistas, disparadores e índices en una muy potente y fácil de usar interfaz gráfica de usuario. El Built-

in Lua motor lenguaje de scripting de SQLiteManager es lo suficientemente flexible para que pueda generar informes o interactuar con bases de datos SQLite en casi cualquier manera. (SQLite Manager, 2010)

1.6 Conclusiones parciales

A partir del análisis de las soluciones existentes para el mapeo objeto-relacional, se verificó la necesidad de realizar un componente que pueda ser integrado al sistema Syam. Identificándose los principales elementos a tener en cuenta para el diseño e implementación de la solución propuesta.

A su vez después de analizar las políticas y normativas arquitectónicas del proyecto Sistema Minero Cubano y las particularidades de la propuesta de solución se decide utilizar como metodología de desarrollo AUP, visual Paradigm 8.0 como herramienta de modelado y como lenguaje de modelado UML 2.0. Durante la implementación de la solución se hará uso del IDE de desarrollo QtCreator en su versión 5.2, utilizando como lenguajes de programación C++, soportado por el marco de trabajos Qt .Además se usará como gestores de base de datos PostgreSQL 9.4 y SQLite 3, para la administración de la base de datos PGAdmin 9.3 y SQLite Manager 1.2.

Capítulo 2: Análisis y diseño del componente para el mapeo objeto-relacional del sistema Syam

2.1 Introducción

En este capítulo se realiza la descripción del componente a desarrollar. Para lograr una mayor comprensión por parte de los desarrolladores de las clases conceptuales significativas asociadas al dominio del problema. El funcionamiento del sistema y los procesos de negocio, se muestran con el diagrama de Modelo de Dominio. Además de generar los diagramas de clases, diagramas de caso de uso y sus descripciones, así como los requisitos funcionales y no funcionales que se proponen para el funcionamiento del Componente mapeo objeto-relacional para el sistema Syam. Los artefactos y documentación relacionados con los aspectos mencionados serán realizados siguiendo las pautas de la metodología de desarrollo de software AUP, seleccionada en el capítulo anterior.

2.2 Modelo de Dominio del componente

El Modelo de Dominio, en lo adelante MD, consiste en uno o más diagramas de clases que utilizando el lenguaje UML, muestra los conceptos básicos del dominio del problema, sus propiedades más importantes, además de las relaciones más significativas entre dichos conceptos. Un MD es una representación de las clases conceptuales del mundo real. Junto con los requerimientos constituye la entrada más importante para el diseño de un sistema, además de permitir establecer los conceptos y términos relacionados con el problema y validar la comprensión de los desarrolladores. A continuación se muestra el modelo de dominio del componente para el mapeo objeto-relacional.

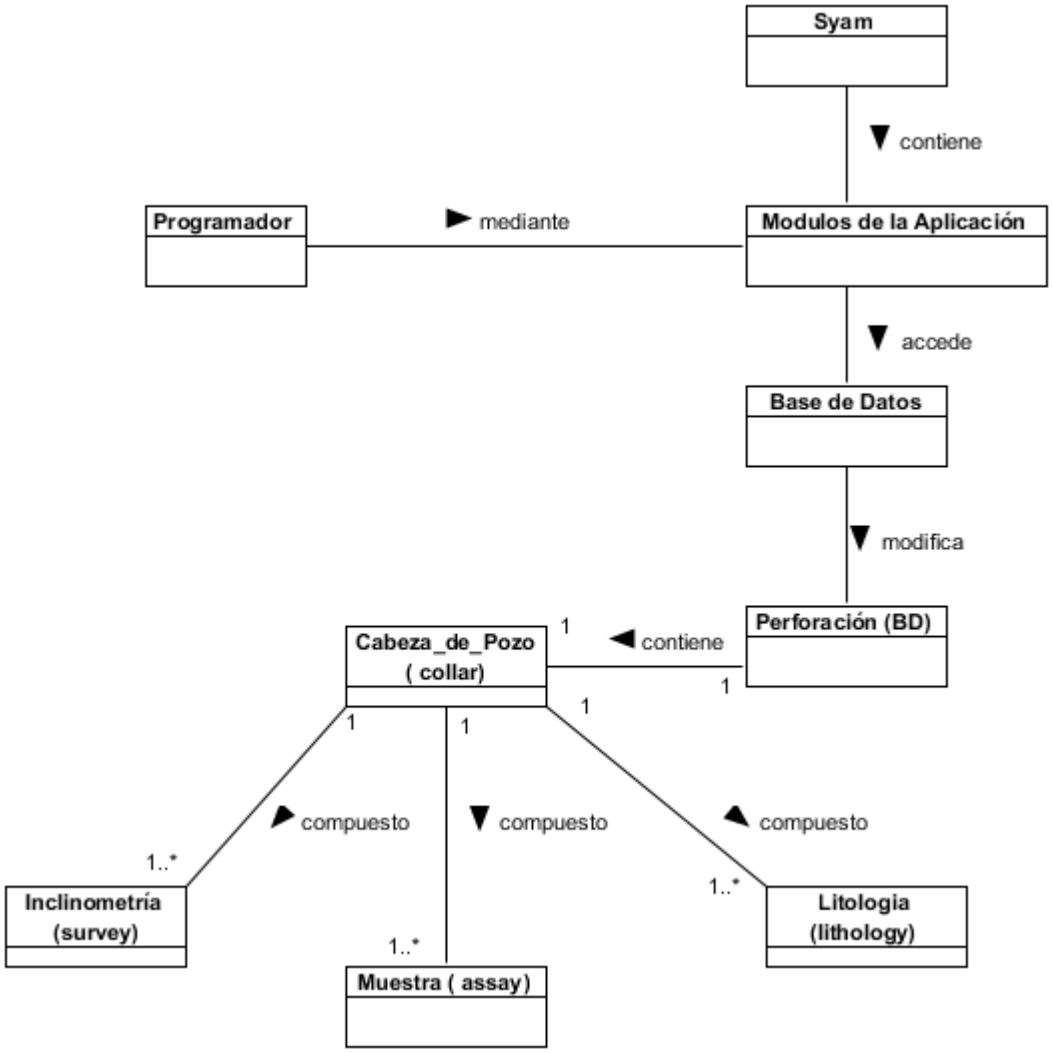


Figura 1: Diagrama de Modelo de Dominio

2.2.1 Descripción de las clases del Modelo de Dominio

Syam: Es el sistema, el cual contiene el módulo de visualización con el cual se accede a las bases de dato.

Programador: Es quien realiza las consultas SQL para la obtención de los datos almacenados en las BD, accediendo a estos mediante el módulo de visualización, desde cualquier capa de la aplicación que sea necesario.

Módulos de la Aplicación: Cuenta con los módulos: Bases de datos, Visualización y Núcleo del sistema.

Bases de Datos: Contienen las tablas que almacenan la información con que trabaja el sistema.

Perforación: Es la alusión a una base de datos la cual está conformada por el conjunto de datos recogidos en los diferentes ficheros que se almacenan en las siguientes tablas: tb_collar, tb_survey, tb_assay, tb_lithology. Un nuevo proyecto del sistema es una perforación que almacena los datos en dichas tablas.

Cabeza de pozo: Tabla en la BD que almacena los valores asociados a la ubicación exacta en el terreno de un pozo perforado, mediante las coordenadas x, y, z.

Inclinometría: Tabla en la BD que almacena los valores asociados al ángulo de inclinación de un pozo.

Muestra: Tabla en la BD que almacena la concentración de minerales por cada muestra encontrada en un pozo, además de los intervalos en que aparecen, atendiendo a los pozos a que pertenecen.

Litología: Tabla en la BD que almacena las diferentes capas litológicas, las cuales se refieren a los tipos de roca que contiene cada pozo.

2.4 Requisitos del componente para el mapeo objeto-relacional en el sistema Syam

Como parte del proceso de desarrollo, y para lograr abarcar todas las funcionalidades necesarias para el correcto funcionamiento del componente mapeo objeto-relacional para el sistema Syam. Se hace necesario la extracción de los requerimientos con los cuales debe cumplir dicho componente.

"Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. En el otro extremo, es una definición detallada y formal de una función del sistema." (Sommerville, 2005)

2.4.1 Requisitos Funcionales

Para desarrollar un producto informático es necesario conocer cuáles son las funciones que debe cumplir, además de esclarecer cuáles son sus principales características.

"Los Requerimientos Funcionales son condiciones o capacidades que el sistema debe cumplir, suficientemente buenas como para llegar a un acuerdo entre los clientes (incluyendo usuarios) sobre qué debe y qué no debe hacer el sistema." (Jacobson, y otros, 2000)

Teniendo en cuenta los patrones arquitectónicos a aplicar, los cuales se especifican más adelante, es necesario esclarecer en la siguiente tabla la correspondencia entre los componentes de la BD y la POO para un mejor entendimiento de los requisitos funcionales:

Tabla 2: Correspondencia entre componentes de la BD y la POO

Sistema Gestor de Base de Dato	Lenguaje de Programación
Celda	Se interpreta como un **atributo o propiedad** de un objeto.
Tupla	Se interpreta como un **objeto de negocio**.
Tabla	Se interpreta como una colección (**repositorio)** de objetos de negocio.

A continuación se muestran los requisitos funcionales del componente a desarrollar:

RF1: Establecer conexión al servidor de base de datos PostgreSQL.

Descripción: Permitirá al seleccionar el SGBD PostgreSQL y establecer la conexión mediante los parámetros correspondientes, para manejar el acceso y uso de las BD que contiene.

Entrada: SGBD PostgreSQL (seleccionado) y parámetros de conexión: usuario, contraseña, servidor, puerto.

Salida: Conexión con el SGBD PostgreSQL establecida.

RF2: Listar base de datos existentes en el SGBD PostgreSQL.

Descripción: Después de establecida la conexión con el SGBD, muestra el listado de nombres de las BD existentes en PostgreSQL.

Entrada: Conexión con el SGBD PostgreSQL.

Salida: Lista de los nombres de las BD existentes en PostgreSQL.

RF3: Permitir conexión con una BD en PostgreSQL.

Descripción: Permite la conexión a una BD existente en el SGBD PostgreSQL, mediante la selección del nombre de la BD a la que se quiere conectar.

Entrada: Nombre de la BD a conectar.

Salida: Conexión con la BD establecida.

RF4: Permitir conexión con una BD en SQLite.

Descripción: Permitirá al seleccionar el SGBD SQLite, la conexión a una BD existente en el SGBD SQLite, mediante el nombre de la BD a la que se quiere conectar.

Entrada: Nombre de la BD a conectar.

Salida: Conexión con la BD establecida.

RF5: Transformar una celda de una tabla en atributo de un objeto de negocio.

Descripción: Obtiene un atributo en un objeto de negocio. Interpretando una celda de una tabla de la BD en un atributo o propiedad de un objeto de negocio.

Entrada: Celda a convertir en atributo o propiedad.

Salida: Atributo o propiedad de un objeto de negocio.

RF6: Transformar tupla de una tabla de la BD en un objeto de negocio.

Descripción: Crea un objeto de negocio a partir de una tupla o fila de una tabla de la BD. Donde cada objeto de negocio creado contiene atributos o propiedades que equivalen a las celdas que contiene la tupla en la BD.

Entrada: Tupla de la BD a convertir en objeto de negocio.

Salida: Objeto de negocio creado sobre una tupla de la base de datos.

RF7: Transformar tabla de la BD en repositorio contenedor de objetos de negocio.

Descripción: Crea un repositorio que contiene objetos de negocio con sus atributos, equivalente a una tabla de la BD con sus tuplas y celdas.

Entrada: Tabla de la BD.

Salida: Repositorio contenedor de objetos de negocio.

RF8: Insertar una tupla de una tabla de la BD a partir de un objeto de negocio.

Descripción: Al insertar un objeto de negocio en el sistema y salvar los cambios realizados, los datos contenidos en el objeto son almacenados en la BD como tuplas de la tabla correspondiente a él.

Entrada: Objeto de negocio a insertar.

Salida: inserción de una tupla en una tabla de la BD.

RF9: Actualizar tupla de una tabla en la BD.

Descripción: Se actualiza una tupla de una tabla de la BD mediante el objeto de negocio correspondiente a esta, consiguiendo que los cambios realizados en el objeto de negocio también sean realizados sobre la tupla a la cual hace referencia en la tabla de la BD.

Entrada: Objeto de negocio actualizado.

Salida: Tupla de una tabla de la BD actualizada.

RF10: Eliminar una tupla de una tabla de la BD.

Descripción: mediante un objeto de negocio, elimina la tupla equivalente al mismo en la tabla de la BD.

Entrada: Objeto de negocio a eliminar.

Salida: Tupla eliminada de una tabla de la BD.

RF11: Insertar un atributo a un objeto de negocio como una columna de una tabla de la BD.

Descripción: Tiene como objetivo añadirle a una tabla de la BD una columna, a partir de un atributo de un objeto de negocio, equivalente a dicha columna.

Entrada: Atributo a añadir.

Salida: Columna añadida en una tabla de la BD.

RF12: Cerrar conexión con el SGBD.

Descripción: Termina la conexión con el SGBD en uso, terminando así el acceso a las BD del mismo y a la creación de nuevas BD.

Entrada: Solicitud de cierre de conexión.

Salida: SGBD desconectado.

2.4.2 Requisitos no funcionales

"Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener, además son aspectos importantes que el producto debe cumplir para lograr un producto atractivo, usable, rápido o confiable." (Pressman, 2005) A continuación se muestran los requerimientos no funcionales que debe cumplir el componente para su perfecto funcionamiento:

RNF: Hardware.

Como el componente a desarrollar se integra al sistema Syam, este tendrá que asumir los mismos requerimientos que en cuanto a hardware tiene el sistema.

Servidor de BD:

- Memoria RAM de 2GB.
- Procesador superior al Intel Pentium IV de 2.8 GHz.
- Con soporte múltiples núcleos o 1 núcleo.

En las PC clientes:

- Memoria RAM de 512 MB.
- Procesador superior o igual al Intel Pentium IV de 2.8 GHz.

RNF: Soporte.

El Componente para el mapeo objeto-relacional debe estar codificado bajo los estándares de BD de SQL99 y el estándar de codificación para C++, establecidos ambos en el proyecto sistema minero cubano al cual pertenece el sistema Syam, que será donde se integre el componente.

RNF: Software.

El servidor a utilizar es el mismo que utiliza Syam, ya que el componente funciona como parte del sistema:

- Sistema Operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 8.04 hasta 14.04, Debian 4.0 GNU/Linux.

2.5 Descripción del componente propuesto

Con el objetivo de cumplir con la propuesta planteada para dar solución a la problemática, se hace necesario la confección del modelo de casos de uso. El cual permite un buen entendimiento del sistema por parte de los desarrolladores para cumplir con todas las necesidades de Syam. Mediante su estructura de actores, casos de usos y sus relaciones se describe de modo claro el funcionamiento del sistema.

2.5.1 Definición de los casos de uso

Un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. El modelo de caso de uso del sistema contiene actores, casos de uso y sus relaciones. Describe lo que hace el sistema para cada tipo de usuario, representados por uno o varios actores, los cuales representan a individuos o sistemas externos que colaboran con este.
Para desarrollar y construir un software es útil apoyarse en la experiencia que brindan las soluciones ya implementadas en proyectos semejantes. Se tendrán identificados los problemas potenciales, conociendo que la solución va a funcionar. Y se pueden catalogar estas soluciones a problemas comunes, como patrones, y son utilizados en diversos aspectos del desarrollo de software.
Un patrón de casos de uso no describe un uso particular de un sistema. Más bien, captura técnicas para que el modelo sea mantenible, reusable, y entendible. La aplicación de patrones de casos de uso trae beneficios como aumentar la productividad, reutilización de elementos existentes además de no invertir tiempo en resolver problemas ya resueltos.

Como patrón de Caso de Uso, es válido aplicar el patrón CRUD por sus siglas en inglés Create, Read, Update and Delete (Crear, Obtener, Actualizar y Borrar), que propone formar un caso de uso a partir de los requisitos funcionales relacionados con las acciones de insertar, listar o mostrar, modificar, y eliminar una determinada información. Pero atendiendo a que todas estas acciones no se encuentran reflejas en los

requisitos funcionales, se adoptará la utilización del patrón CRUD parcial. Haciendo uso de este, las funcionalidades identificadas para el desarrollo de la aplicación quedaron agrupadas en cuatro casos de uso:

- Conectar a una Bases de Datos.
- Administrar tupla como objeto de negocio.
- Administrar columna como atributo.
- Convertir tablas a repositorios.

2.5.2 Descripción de los actores

Tabla 3: Descripción de los actores

Actor del Sistema	Descripción
Syam	El sistema Syam es el encargado de realizar las operaciones sobre la base de datos, por medio del componente para el mapeo objeto-relacional que forma parte de la capa de acceso a datos. Brindándole al desarrollador una abstracción del gestor de base de datos a utilizar, evitándole la constante interacción con las sentencias SQL para acceder a los datos.

2.5.3 Diagrama de Casos de Uso del Sistema

A continuación se presenta el diagrama de caso de usos del sistema**:**

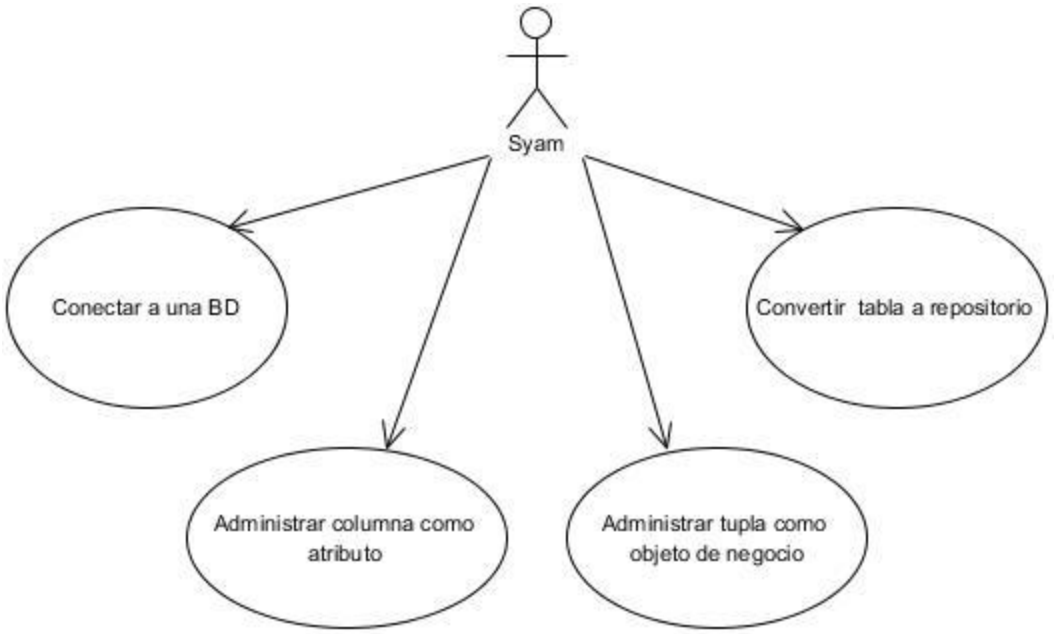


Figura 2: Diagrama de casos de uso

2.5.4 Especificación de casos de uso

CU 3: Administrar tupla como objeto de negocio

Tabla 4: Descripción de caso de uso Administrar tupla como objeto de negocio

Caso de Uso:	Administrar tupla como objeto de negocio
Descripción:	El CU permite convertir una tupla de una tabla en un objeto de negocio. Además de añadir, modificar o eliminar una tupla mediante un objeto de negocio.
Actores:	Syam
Precondiciones:	Ninguna
Referencias:	RF: 6,8,9,10
Prioridad:	Alta
Curso Normal de Eventos.	

Acción del actor:	Respuesta del sistema:
El CU inicia cuando el actor envía una solicitud al sistema para ejecutar una de las siguientes funcionalidades: Adicionar objeto de negocio. Eliminar objeto de negocio. Modificar objeto de negocio. Importar datos.	
	2: Si el actor ejecuta “Adicionar objeto de negocio” el sistema añade un objeto de negocio en el repositorio correspondiente a este objeto. Ver: Sección “Adicionar objeto de negocio”. Si el actor ejecuta “Eliminar objeto de negocio”, el sistema elimina el objeto de negocio solicitado por el actor. Ver: Sección “Eliminar objeto de negocio”. Si el actor ejecuta “Modificar objeto de negocio”, el sistema modifica el objeto de negocio solicitado por el actor con los datos que este brinda. Ver: Sección “Modificar objeto de negocio”. Si el actor ejecuta “Importar datos” el sistema almacena como tuplas en la BD los objetos de negocio importados. Ver: Sección “Importar datos.”
Sección “Adicionar objeto de negocio”. Flujo Normal de Eventos.	
Acción del actor:	**Respuesta del sistema:**
1: El actor captura los datos a adicionar.	
2: El actor crea un objeto de negocio con los datos capturados.	
3: El actor ejecuta la funcionalidad “Adicionar objeto de negocio”	4: El sistema captura el objeto de negocio a insertar.

	5: El sistema crea objetos del negocio a partir de los datos capturados.
	6: Termina el CU.
Sección "Eliminar objeto de negocio". Flujo Normal de Eventos.	
1:El actor ejecuta la funcionalidad "Eliminar objeto de negocio"	2: El sistema captura el identificador del objeto de negocio a eliminar.
	3: El sistema crea un filtro con el identificador capturado.
	4: El sistema selecciona el objeto de negocio que se desea eliminar utilizando el filtro creado.
	5: El sistema elimina el objeto de negocio de su respectivo repositorio.
Sección "Modificar objeto de negocio". Flujo Normal de Eventos.	
1: El actor ejecuta la funcionalidad "Modificar objeto de negocio"	2: El sistema prepara las sentencias SQL que contienen los cambios a realizar.
	3: El sistema guarda en la BD los cambios realizados en los objetos de negocio.
	4: Termina el CU.
Sección "Importar datos". Flujo Normal de Eventos.	
1: El actor ejecuta la funcionalidad "Importar datos"	2: El sistema captura los datos a importar.
	3: El sistema adiciona objetos de negocio a sus respectivos repositorios utilizando los datos importados. Ir a la sección "Adicionar objeto de negocio".
	4: El sistema guarda como tuplas en la BD los objetos de negocio importados.
	5: Termina el CU.

2.7 Arquitectura del sistema

La Arquitectura de Software consiste básicamente en un conjunto de abstracciones y patrones relacionados entre sí que brindan el marco de referencia necesario para orientar y dirigir la línea de construcción del software. Se diseña y selecciona basándose en restricciones y objetivos bien definidos. Define de manera abstracta los componentes que llevan a cabo algunas tareas, sus interfaces y la comunicación entre ellos.

Con la definición que propone la IEEE 1471-2000 queda sintetizado con claridad el significado de Arquitectura de software: "*… es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución".*

Pero algunos estudiosos de esta materia también han hecho sus aportes, tal es el caso de Paul Clements: *"La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones".* (Clements, 1996)

2.7.1 Estilos arquitectónicos

Los estilos arquitectónicos son fundamentales para dar forma a la arquitectura de una aplicación, son principios que definen a alto nivel un aspecto de la aplicación. Un estilo arquitectónico se define por un conjunto de componentes, un conjunto de conexiones entre dichos componentes y un conjunto de restricciones sobre cómo se comunican dos componentes cualesquiera conectados.

En el componente para el mapeo objeto-relacional a desarrollar se hace uso una arquitectura **Basada en Componentes**, la cual permite la reutilización de piezas de código preelaborado para la realización de diversas tareas. Los componentes encapsulan alguna funcionalidad a través de interfaces, además de ser combinados en las aplicaciones para realizar una tarea. Es un estilo para diseñar aplicaciones a partir de componentes individuales.

Un componente es diseñado para operar en diferentes entornos y contextos. Toda la información debe ser pasada al componente en lugar de incluirla en él, o que este acceda a ella. Su diseño permite que sean lo más independiente posible de otros componentes, por lo que puede ser desplegado sin afectar a otros componentes o sistemas.

Entre las principales características que brinda la arquitectura Basada en Componentes se encuentran: (Buschmann, y otros, 2009)

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.
- Fácil despliegue ya que se puede sustituir un componente por su nueva versión sin afectar al sistema o a otros componentes.
- Pueden ser reutilizados, debido a su independencia de contexto pueden ser utilizados en otras aplicaciones y sistemas.
- Contribuyen a reducir los costes, ya que pueden utilizarse componentes de terceros para abaratar los costes de desarrollo y mantenimiento.
- Mayor calidad, dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

Los beneficios que brinda la arquitectura Basada en Componentes: (Buschmann, y otros, 2009)

- Facilidad de Instalación. Cuando una nueva versión esté disponible, usted podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.
- Costos reducidos. El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.
- Reusable. El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.
- Facilidad de desarrollo**.** Los componentes implementan un interface bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.

2.7.2 Patrones arquitectónicos

Los patrones de arquitectónicos ofrecen soluciones a problemas de ingeniería y arquitectura de software, brindando una organización estructural esencial para un sistema en desarrollo. Muestran una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados. Presentan un nivel de abstracción mayor que los patrones de diseño, y no deben confundirse en ningún momento con estos.

Como parte de la arquitectura del sistema se tienen en cuanta el uso de los siguientes patrones:

Repository (repositorio): Permite trabajar de una forma simétrica la orientación a objetos con los modelos relacionales, y logra de forma sencilla que las capas de datos puedan ser probadas o *testables*. Por tanto para cada tipo de objeto que necesite acceso global se debe crear un objeto "Repositorio", el cual deberá dar la apariencia de una colección en memoria de todos los objetos de ese tipo. Los repositorios solo se deben definir para las entidades lógicas principales y no para cualquier tabla de la fuente de datos. Contribuye a que en las capas superiores se mantenga focalizado el desarrollo en modelo, delegando todo el acceso y persistencia de objetos a los repositorios.

Sus principales ventajas son:

- Se presenta al desarrollador de la capa de dominio un modelo más sencillo para obtener objetos/entidades persistidos y gestionar su ciclo de vida.
- Desacopla la capa de Dominio y de Aplicación de la tecnología de persistencia, estrategias de múltiples bases de datos, o incluso múltiples fuentes de datos. (de la Torre, y otros, 2010)

El patrón arquitectónico Repository queda evidenciado en las clases: *smCollarRepository, smSurveyRepository, smAssayRepository* y *smLithologyRepository*.

Active Record: Es un patrón en el cual, el objeto contiene los datos que representan a una tupla (o fila) de nuestra tabla o vista, además de encapsular la lógica necesaria para acceder a la base de datos. De esta forma el acceso a datos se presenta de manera uniforme a través de la aplicación (lógica de negocio + acceso a datos en una misma clase). Una clase Active Record consiste en el conjunto de propiedades que representa las columnas de la tabla más los típicos métodos de acceso como las operaciones CRUD (Create, Read, Update, Delete), búsqueda, validaciones, y métodos de negocio. (de la Torre, y otros, 2010)

Se evidencia el uso del patrón Active Record en las clases: *smCollar, smSurvey, smAssay* y *smLithology*.

Data Mapper: El patrón mapeo de datos tiene como objetivo separar las estructuras de los objetos de las estructuras de los modelos relacionales, y realizar la transferencia de datos entre ambos. Al usar Data Mapper los objetos pueden ignorar el esquema presente en la BD, por lo que no es necesario que hagan uso de lenguaje SQL. (Flower, 2003)

El patrón Data Mapper se evidencia en las clases: *smPgsqlDriver* y *smSqliteDriver*.

Table Data Gateway: Este patrón significa Puerta de enlace a tabla de datos. Donde un objeto actúa como una puerta de enlace a una tabla de base de datos. Mezclar lenguaje SQL en la lógica de aplicación puede causar varios problemas y los administradores de bases de datos tienen que ser capaces de encontrar SQL fácilmente para que puedan encontrar la manera de ajustar y desarrollar la base de datos. Una puerta de enlace a una tabla de datos tiene todo el SQL para acceder a una sola tabla o vista: selecciona, inserciones, actualizaciones y eliminaciones. Otro código llama a sus métodos para toda la interacción con la base de datos. (Flower, 2003)

Este patrón esta evidenciado en las clases *smPgsqlDriver* y *smSqliteDriver*.

Query Object: Un objeto que representa una consulta de base de datos. El lenguaje SQL puede ser complicado, y muchos desarrolladores no están particularmente familiarizados con este. Además, se necesita conocer el esquema de base de datos. Esto se puede evitar mediante la creación de métodos de búsqueda especializados que ocultan el SQL dentro métodos parametrizados, pero eso hace que sea difícil para formar consultas. Un objeto de consulta es un intérprete, es decir, una estructura de objetos que se pueden formar en sí en una consulta SQL. Puede crear esta consulta junto a las clases y los campos en lugar de tablas y columnas. De esta manera, los que escriben las consultas pueden hacerlo independientemente del esquema de base de datos y cambios en el esquema puede ser localizado en un solo lugar. (Flower, 2003)

Se ve evidenciado este patrón en las clases: *smPqsqlCollection* y *smSqliteCollection*.

2.8 Patrones de diseño

Al diseñar un sistema es importante seguir patrones: “Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema general de diseño en un contexto particular.” (Küng, y otros, 2006)

Mediante el uso de patrones, las soluciones a problemas que pueden surgir durante el desarrollo de un sistema, pueden ser reutilizadas tomando como referencia un problema similar resuelto. Lo cual permite un vocabulario común entre diseñadores, además de estandarizar el modo en que se realiza el diseño.

Patrones GRASP: Los patrones GRASP (Patrones de Software para la asignación General de Responsabilidad) constituyen un apoyo para la enseñanza, pues ayudan a entender el diseño de objetos. Según especialistas en esta materia: *“Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable”*. (Grosso, 2011)

De los diferentes patrones que ofrece GRASP para la modelación del componente se han tenido en cuenta los siguientes:

Creador: El patrón Creador guía la asignación de responsabilidades y ayuda a identificar quién debe ser el responsable de la creación de objetos. Se asigna la tarea a una clase de crear cuando, contiene, agrega, compone, almacena o usa otra clase. El propósito fundamental de este patrón es encontrar un creador que se deba conectar con el objeto producido en cualquier evento. El patrón de diseño explicado, se evidencia dentro de la estructura de diseño del componente en las clases *smCollarRepository, smAssayRepository, smSurveyRepository y smLithologyRepository*:

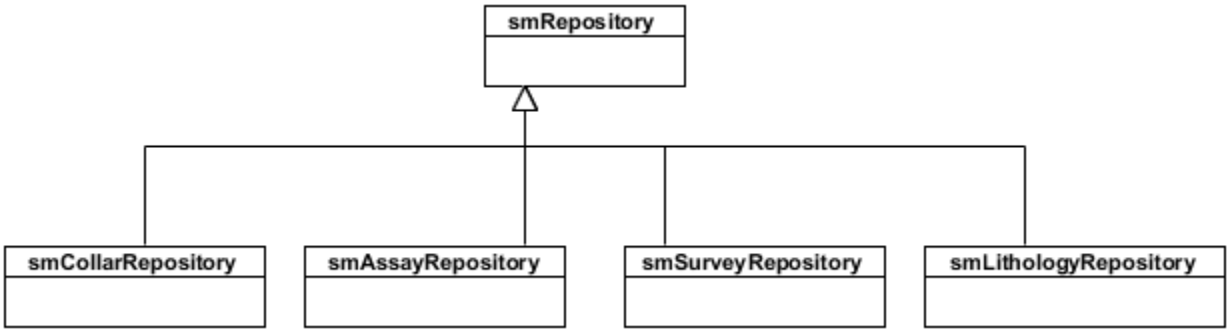


Figura 3: Clases donde se evidencia el uso del patrón GRASP: Creador

Bajo Acoplamiento: Evita la dependencia entre las clases, evitando que las clases estén ligadas entre sí, para que en caso de que una clase sea modificada, tenga la mínima repercusión en el resto de las clases, además de aumentar la reutilización. Este bajo acoplamiento se evidencia clase *smManager*:

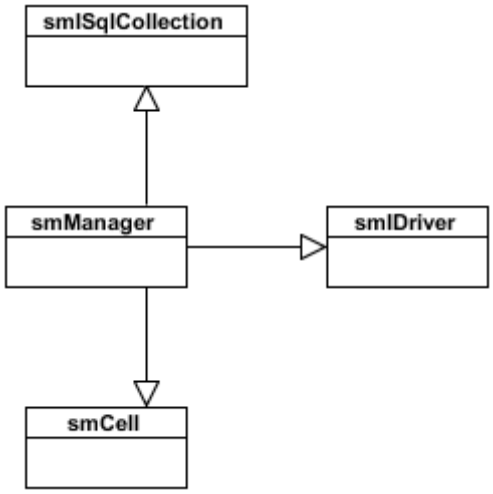


Figura 4: Fragmento del diagrama de clases donde se evidencia el uso del patrón GRASP: Bajo Acoplamiento

Alta cohesión: Asigna responsabilidades de manera que la información que almacena una clase sea coherente y esté relacionada con la clase. La cohesión, son medidas de cuán relacionadas y enfocadas están las responsabilidades de una clase. Las clases con baja cohesión son difícil de reutilizar. Una alta cohesión funcional según Grady Booch: *se da cuando los elementos de un componente colaboran para producir algún comportamiento bien definido.* (Booch, 1994) La alta cohesión esta evidenciada en las clases: *smISqlCollection*, *smManager*, *smIDriver* y *smCell:*

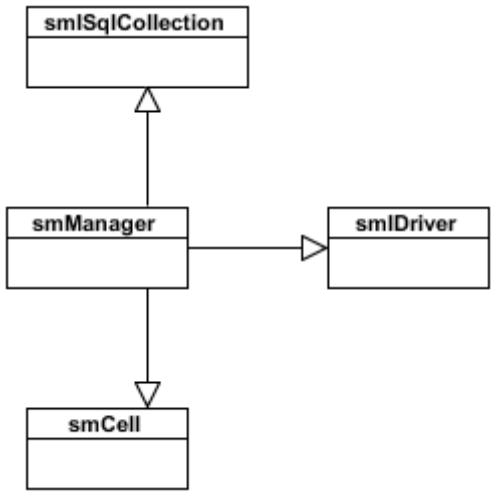


Figura 5: Fragmento del diagrama de clases donde se evidencia el uso del patrón GRASP: Alta cohesión

Experto: Es la forma de saber qué responsabilidad dar a cada objeto. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. Manteniendo la información encapsulada (disminuye el acoplamiento), y obteniendo un diseño con mayor cohesión. Se ve reflejado su uso en la clase *smEntity:*

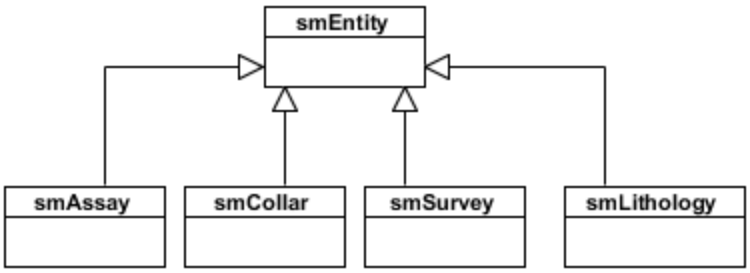


Figura 6: Fragmento del diagrama de clases donde se evidencia el uso del patrón GRASP: Experto

Controlador: Un Controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación. El uso de este patrón está representado en la clase *DataAccess:*

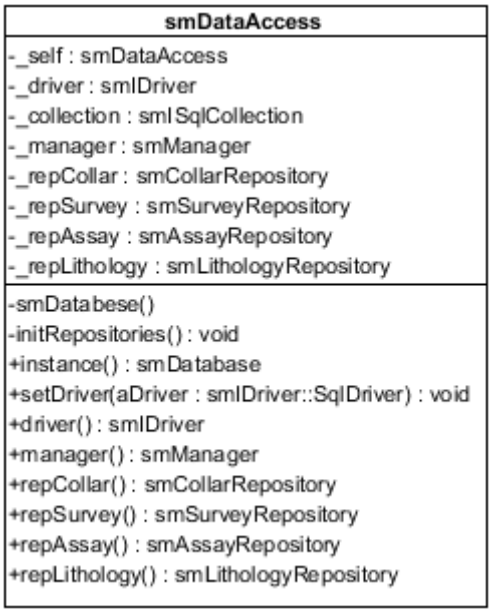


Figura 7: Clase donde se evidencia el uso del patrón GRASP: Controlador

Patrones GOF: Del inglés (*Gang of Four*, "Pandilla de los Cuatro") agrupan en tres categorías: de creación, de estructura y de comportamiento, una serie de patrones de diseño recopilados. Y en lo descrito por Roger Pressman: *"están dirigidos al desarrollo de sistemas orientados a objetos y se encuentran divididos en tres grupos: los de creación utilizados en la abstracción de cómo es creado un objeto, los de estructura que indican cómo se encuentran compuestas las clases y los de comportamiento utilizados en la asignación de responsabilidades en las clases".* (Pressman, 2005)

Los patrones GOF utilizados para el desarrollo del componente para el mapeo objeto-relacional para el sistema Syam fueron:

Facade (Fachada): Este patrón simplifica el acceso a un conjunto de clases proporcionando una única clase que todos utilizan para comunicarse con dicho conjunto de clases. Los clientes no necesitan conocer las clases que hay tras la clase *facade.* Además se pueden cambiar las clases "ocultadas" sin necesidad de cambiar los clientes. Sólo hay que realizar los cambios necesarios en *facade.* Este patrón de diseño esta evidenciado en la clase *smRepository* y *smEntity.* (Gamma, y otros, 2004)

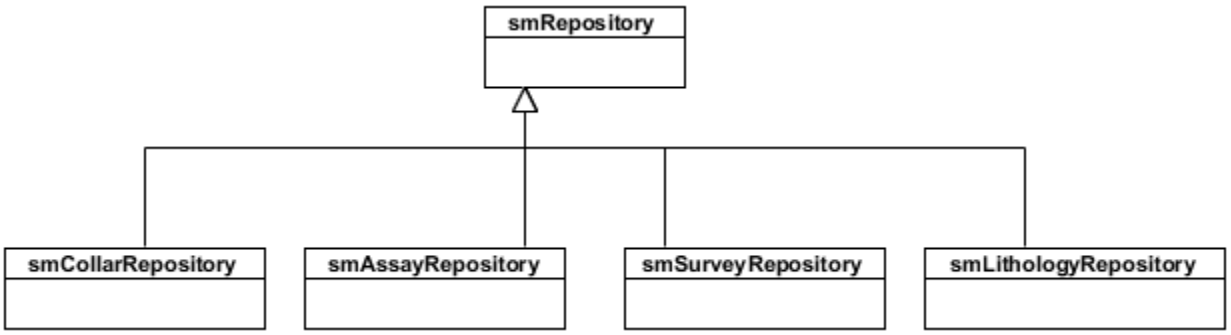


Figura 8: Fragmento del diagrama de clases donde se evidencia el uso del patrón GoF: Facade

Template Method (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura. Tiene la intención de proporcionar un método que permite que las subclases redefinan parte del método sin rescribirlo. Este patrón de diseño se pone de manifiesto en las clases *smIDriver* y las relacionadas con esta *smPgsqlDriver* y *smSqliteDriver*, además ser utilizado en las clases correspondientes a los repositorios. (Gamma, y otros, 2004)

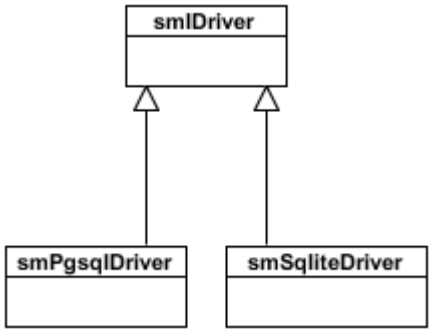


Figura 9: Fragmento del diagrama de clases donde se evidencia el uso del patrón GoF: Template Method

2.9 Modelo de Diseño

El modelo de diseño es un modelo físico y concreto, da forma al sistema y debe ser mantenido durante todo el desarrollo del ciclo de vida del software. Se centra en la realización de los casos de uso a través de los requisitos, tanto funcionales como no funcionales. La entrada esencial a este modelo es el resultado del modelo de análisis, ya que este resultado proporciona una comprensión detallada de los requisitos. El objetivo final del flujo de trabajo de diseño es producir un modelo lógico del sistema a implementar.

2.9.1 Diagrama de clases del diseño

El diagrama de clases del diseño, representa la realización física de los casos de uso. Describe gráficamente las especificaciones de las clases de software y las interfaces y contiene las definiciones de las entidades del software en vez de conceptos del mundo real.

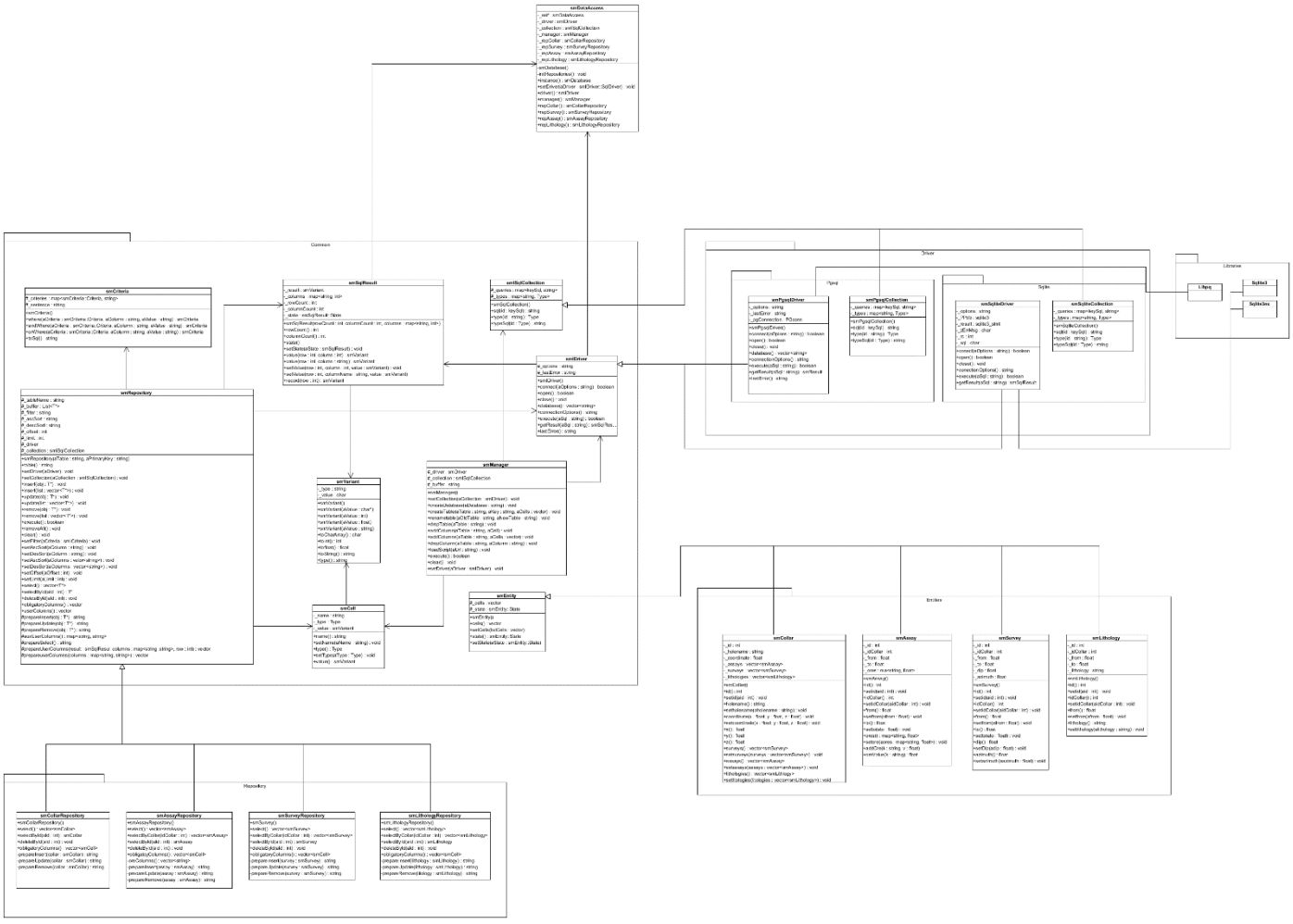


Figura 10: Diagrama de Clases del diseño

2.9 Conclusiones parciales

En este capítulo se realizó la presentación del diseño del componente propuesto, definiendo el comportamiento de este mediante los requisitos funcionales identificados. Con una sencilla agrupación en casos de uso de dichos requisitos se tiene una clara visión de las funcionalidades a implementar.

Con el objetivo de obtener la solución se hizo uso de los artefactos generados por la metodología de desarrollo AUP. Cada artefacto generado contribuyo en el desarrollo del componente permitiendo que los resultados se ajustaran a la solución del problema a resolver. La realización de los diagramas permitió dar una visión clara para dar paso a la etapa de implementación. Identificándose CU mediante los requisitos funcionales antes descritos. Es significativo destacar el uso de patrones, que garantiza la calidad del código fuente, puesto que al ser soluciones a problemas recurrentes, permiten probar el correcto funcionamiento de cada componente, clase o método implementado.

Capítulo 3: Implementación y Pruebas del Componente para el mapeo objeto-relacional en el sistema Syam

3.1 Introducción

En este capítulo se tendrán en cuenta todos los aspectos relacionados con la implementación del Componente para el mapeo objeto-relacional en el sistema Syam, para darle solución a los requerimientos funcionales detallados en el capítulo anterior, y agrupados en casos de usos en el mismo. Además de esclarecer que estándar de codificación se adoptara durante todo el desarrollo, así como que estilos de programación serán seguidos durante toda la implementación. Finalizando con el proceso de pruebas a realizar al componente, para evaluar su desempeño y capacidad de respuesta al integrarse al sistema Syam.

3.2 Modelo de implementación

"El modelo de implementación es una colección de componentes y subsistemas o paquetes que los contienen. Los componentes constituyen una parte física y reemplazable del sistema que cumple y proporciona la realización de un conjunto de interfaces. Estos componentes incluyen: ficheros ejecutables, ficheros de código fuente, y otros tipos de ficheros necesarios para la implementación y el despliegue del sistema. En este modelo se describen las relaciones que existen entre los paquetes y clases del modelo de diseño a subsistemas y componentes físicos". (Jacobson, y otros, 2000)

3.3 Diagramas de secuencia

Los diagramas de secuencia muestran las interacciones entre un conjunto de objetos, ordenadas según el tiempo en que tienen lugar. En un diagrama de secuencia se indican los módulos o clases que forman parte del programa y las llamadas que se hacen en cada uno de ellos para realizar una tarea determinada. Estos son útiles para definir acciones que se puedan realizar en la aplicación. En la siguiente figura se muestra el diagrama de secuencia para la sección Insertar objeto de negocio en el CU Administrar objeto de negocio como tupla. Este ejemplo se corresponde a la inserción de un objeto tipo "collar":

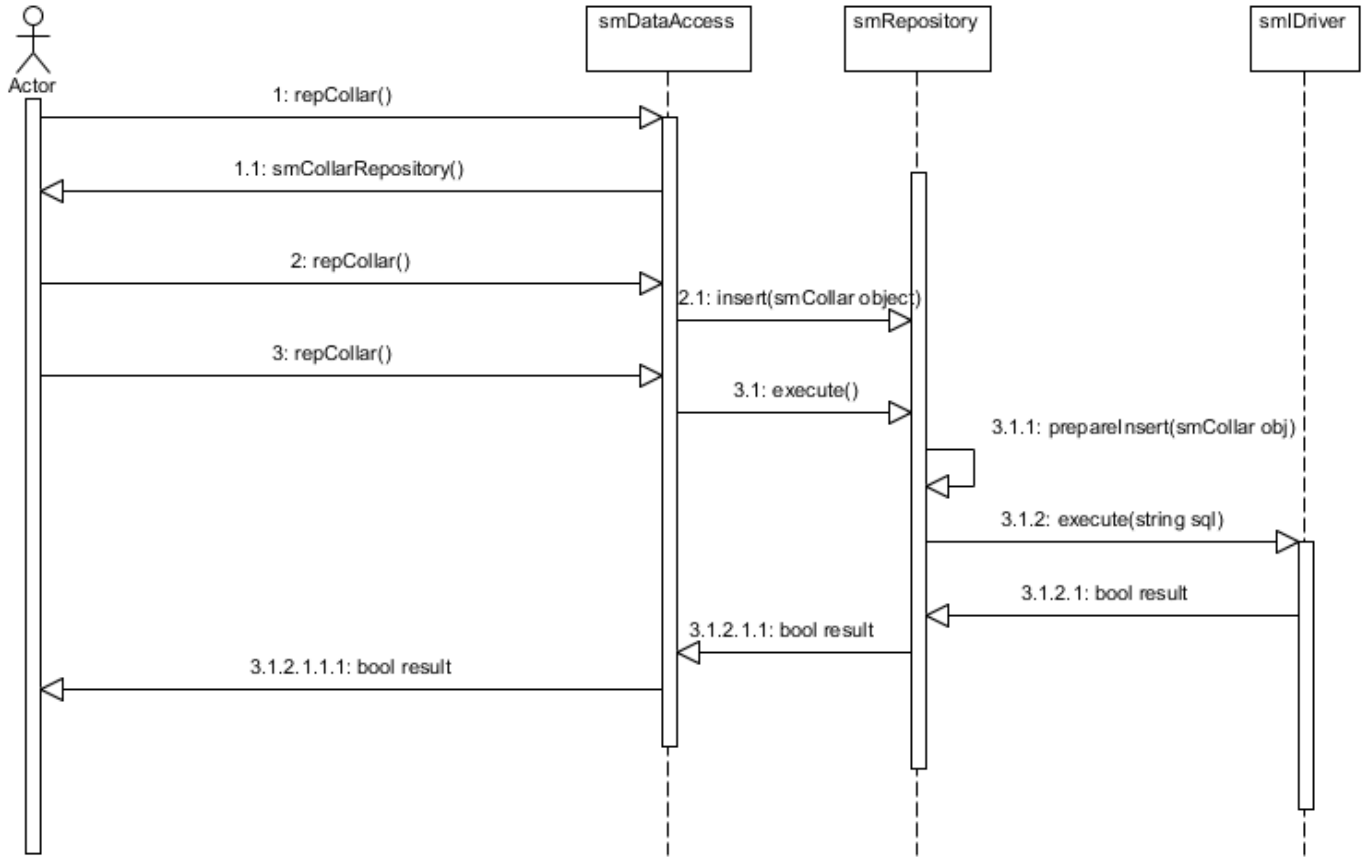


Figura 11: Diagrama de Secuencia CU Administrar objeto de negocio como tupla

Para lograr la creación de objetos de negocio, que contengan los atributos relacionados a las columnas opcionales contenidas en las tablas de la BD, el componente realiza los siguientes pasos:

- Selecciona toda la información de la tabla asociada a los objetos de negocio a crear.
- Selecciona el nombre y el tipo de dato de las columnas adicionales, con el fin de convertir cada columna adicional en un atributo del objeto de negocio a crear.
- Los objetos de negocio son creados realizando un ciclo en el que se agregan a los atributos comunes y los establecidos mediante las columnas adicionales, la información obtenida.

3.4 Diagrama de componente

Los componentes no son más que partes de software que son utilizados a través de interfaces bien definidas, brindando determinados servicios. En el diagrama de componente se encuentran descritas las interacciones que existen entre los componentes principales que intervienen en la solución, ya sean bibliotecas, componentes de código fuente o archivos, conteniendo además sus dependencias más significativas.

La solución propuesta contiene el uso de archivos o clases y bibliotecas, para el manejo de los sistemas gestores de bases de datos PostgreSQL y SQLite. Tal es el caso de la librería Libpq, la cual es un conjunto de librerías escritas en C, que permiten a un programa cliente enviarle consultas al servidor PostgreSQL y recibir el resultado de estas consultas. Y por otra parte para el manejo de las consultas almacenas con SQLite se realiza mediante la importación de tres clases: *sqlite3.c*, *sqlite3.h*, *sqlite3ext.h*. En ambos casos las clases y librería mencionadas forman parte del siguiente diagrama de componentes:

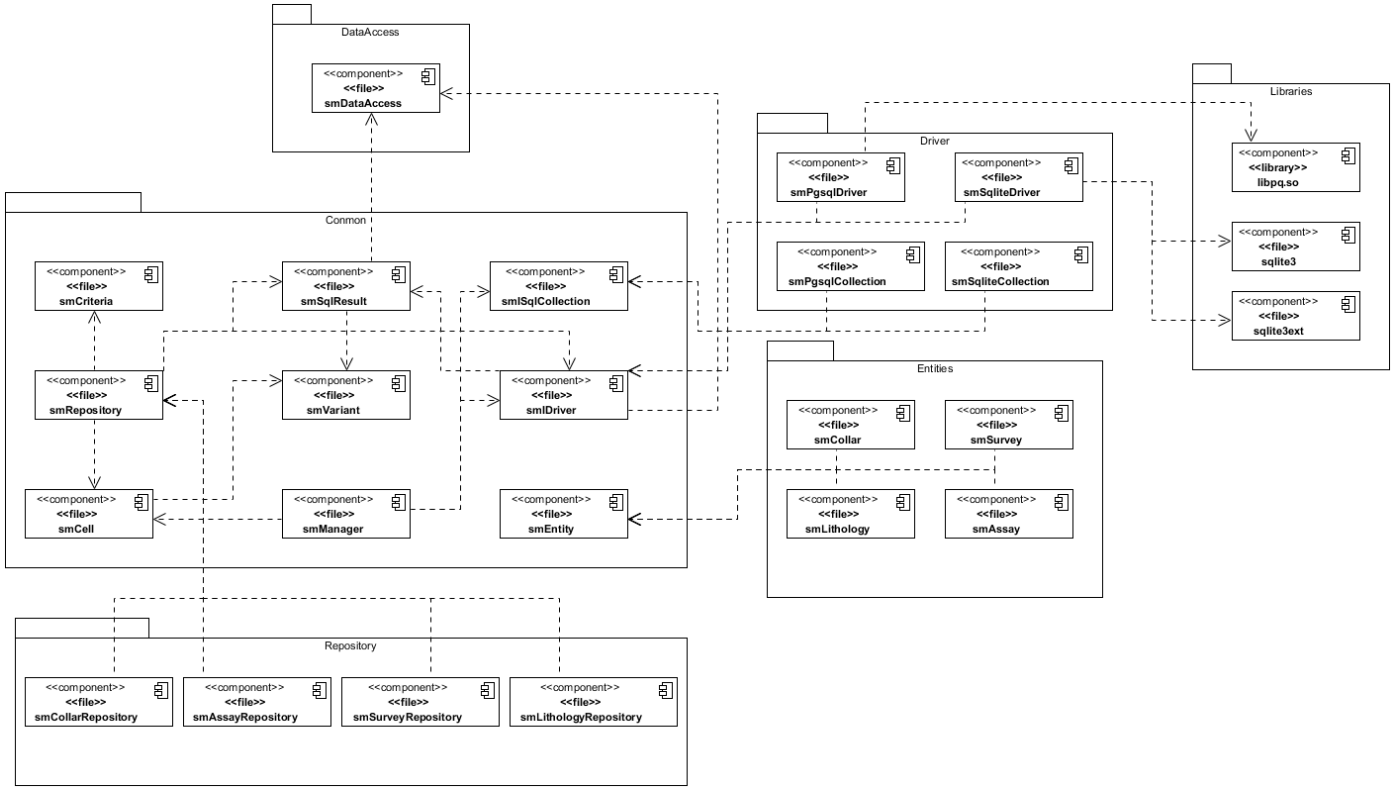


Figura 12: Diagrama de componentes

3.5 Diagrama de despliegue

Los diagramas de despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y la manera en que están repartidos los componentes sobre los nodos. Para el caso del componente para el mapeo objeto-relacional en el que se está trabajando se cuenta con dos nodos. En el nodo Estación de trabajo se deberá contar con un sistema operativo superior a Windows XP o GNU/Linux. Y por su parte el nodo correspondiente a Servidor de base de datos deberá contar con sistema operativo superior Windows XP, o GNU/Linux.

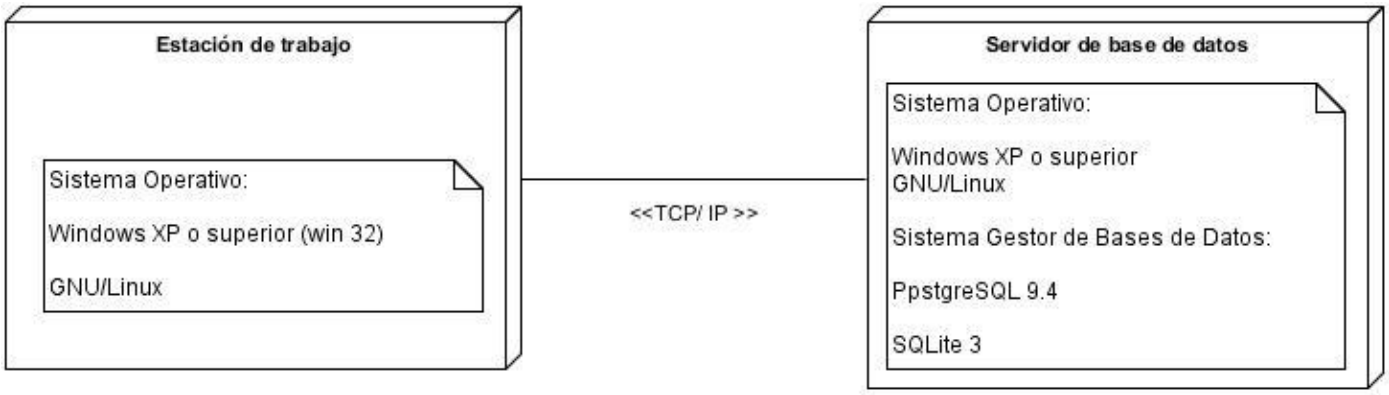


Figura 13: Diagrama de Despliegue

3.6 Estándar de codificación

En cada proyecto de software desde su inicio, sería correcto establecer un estándar de codificación para tener seguro que todos los programadores, trabajan de forma coordinada. Se entiende por estándar de codificación, un estándar que comprenda y controle todos los aspectos que se abarcan en la generación del código. Los programadores son los encargados de la implementación de un estándar de forma prudente, el cual debería tender siempre a lo práctico, con un estilo armonioso para garantizar un código fuente completo. Que un código fuente sea legible incide en la comprensión de un sistema de software por parte de los programadores.

Para la implementación de la solución planteada, es necesario utilizar los estándares de codificación por los que se rige el proyecto Sistema Minero Cubano, en el que se encuentra el sistema Syam del cual formará parte el componente a desarrollar. En el que se tienen en cuenta el manejo de aspectos como:

- Ficheros

- Inclusiones
- Identificadores
- Comentarios
- Organización
- Espacios en blanco
- Llaves
- Clases privadas

3.7 Pruebas de software

En un producto de software son las pruebas el conjunto de técnicas que permiten evaluar la calidad, integrándose estas dentro de las diferentes fases del desarrollo y ciclo de vida del mismo. La calidad de un sistema de software es algo subjetivo, que depende del contexto y del objetivo que se pretenda conseguir con él. Los niveles de calidad son determinados mediante pruebas o medidas que permitan el grado de acatamiento con respecto a las especificaciones iniciales del sistema o mejor dicho, los requisitos funcionales.

Las pruebas de software son la garantía de la calidad de un producto desarrollado, y permiten que sean detectados los errores antes de la entrega al usuario final.

3.7.1 Niveles de prueba

El ámbito de las pruebas de software puede variar en niveles como:

Pruebas Unitarias: Son las encargadas de verificar el funcionamiento aislado de piezas de software que pueden ser probadas de forma separada, ya sean subprogramas/módulos separados, o como componentes que incluyen varios subprogramas/módulos. Estas pruebas suelen realizarse con acceso al código fuente probado, con ayuda de herramientas de depuración, o con la participación opcional de los programadores que desarrollaron el software.

Pruebas de Integración: Verifican la integración de los componentes del sistema de software. Siguiendo como estrategias: La incremental, en la cual se combinan el conjunto de módulos que han sido probados

con el próximo módulo a implementar, es clasificada en incremental ascendente o descendente; o la estrategia guiada por la arquitectura, en la cual los componentes se integran según los hilos de funcionalidad.

Pruebas de Sistema: Son las pruebas que verifican el funcionamiento del sistema en su conjunto. Además de ser las más adecuadas para comprobar los requisitos no funcionales (seguridad, exactitud, velocidad, fiabilidad), ya que en los niveles Unitarias e Integración es donde se suelen detectar los fallos funcionales. Son probadas también en este nivel las interfaces externas con otros sistemas, las utilidades, unidades físicas y el entorno operativo.

Pruebas de Aceptación: En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique. Además de validar los requisitos de rendimiento desde el punto de vista operacional.

De los niveles de prueba citados, se le aplicarán al componente desarrollado pruebas de Sistema. Para así garantizar el perfecto funcionamiento del mismo, y que cumpla con los requerimientos funcionales mencionados en el anterior capítulo.

3.7.2 Métodos de prueba

Los métodos de prueba tienen como objetivo encontrar el mayor número de fallos posibles, intentando ser sistemáticos en identificar un conjunto representativo de comportamientos del programa. Determinados por el dominio de subclases del dominio de entradas, escenarios o estados. En general existen dos enfoques diferentes:

- El funcional, conocido como Caja Negra, donde los casos de prueba se basan en el comportamiento de entrada/salida y se realizan pruebas de forma que se compruebe que cada función es operativa. En esta los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta.
- El estructural, o llamado Caja Blanca, basado en información sobre cómo ha sido diseñado o codificado el software, permitiendo desarrollar pruebas de forma que se asegure que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han probado de forma adecuada. En esta prueba se realiza un examen minucioso de los detalles procedimentales,

comprobando los caminos lógicos del programa, comprobando los bucles y condiciones, y examinado el estado del programa en varios puntos.

En la realización de las pruebas serán utilizados los métodos de Caja Negra a nivel de Sistema, para evaluar el comportamiento del componente mediante una pequeña interfaz.

Dentro del nivel de Sistema también se incluyen las pruebas de Rendimiento, que no son más que: *"pruebas realizadas para determinar cómo un sistema se realiza en términos de capacidad de respuesta y estabilidad bajo una carga de trabajo particular. También puede servir para investigar, medir, validar o verificar otros atributos de calidad del sistema, tales como la escalabilidad, la fiabilidad y el uso de recursos."* (Docsetools, 2015)

3.7.3 Técnicas de prueba

Como técnica de prueba a utilizar mediante el método de Caja Negra, será utilizada la técnica de Partición equivalente. La cual es una de las técnicas más efectivas ya que divide el dominio de entrada de un programa en clases de datos, a partir de las cuales se derivan los casos de prueba. Cada una de estas clases de equivalencia representa a un conjunto de estados válidos o inválidos para las condiciones de entrada.

Como técnica de prueba para medir el Rendimiento, se le realizará al componente Pruebas de Estrés, que: *"se utilizan normalmente para comprender los límites superiores de la capacidad dentro del sistema. Este tipo de examen se hace para determinar la robustez del sistema en términos de carga extrema y ayuda a los administradores de la aplicación para determinar si el sistema funcionará suficientemente si la corriente de carga va muy por encima del máximo esperado."* (Docsetools, 2015)

Para los diferentes métodos de prueba, Caja Blanca y Caja Negra, existen casos de prueba. En el proceso de prueba que se le realiza al sistema se hace uso de los casos de prueba de Caja Negra, de ahí el motivo por el cual se diseña un caso de prueba por cada CU. Es válido aclarar que un caso de prueba se define como un conjunto de acciones con resultados y salidas previstas basadas en los requisitos de especificación del sistema.

Prueba de Caja Negra: El método de prueba Caja Negra se realizará mediante la técnica Camino básico el cual consiste en: Obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida

como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.

El siguiente es un ejemplo de caso de prueba para el CU Conectar a una BD:

Tabla 5: Secciones a probar en el caso de uso Conectar a una BD

Nombre de la sección.	Escenario de la sección.	Descripción de la funcionalidad.
SC1: Conexión con PosgreSQL	EC1.1 Conexión con PostgreSQL.	En este escenario se establece la conexión con una BD en PostgreSQL y sus parámetros necesarios.
	EC1.2: Conexión con PostgreSQL con datos inválidos.	Es la misma operación realizada en EC1.1 pero cuando los parámetros de conexión son inválidos.
	EC1.3: Conexión con PostgreSQL con campos vacíos.	Es la misma operación realizada en EC1.1 para cuando alguno de los campos está vacío.
SC2: Conexión con SQLite.	EC2.1: Conexión con SQLite.	En este escenario se permite la conexión con una BD en el SGBD SQLite.
	EC2.2: Conexión con SQLite errónea.	Es la misma operación que EC2.1 para cuando no sea cargado el archivo de la BD.

A partir de esta descripción se detallan las variables que se encuentras asociadas al caso de uso Conectar a una BD:

Tabla 6: Descripción de variables

No.	Nombre de Campo	Clasificación	Puede ser nulo	Descripción.
1.	Servidor	Campo de texto	no	Dirección IP donde se encuentra el SGBD PostgreSQL.
2.	Puerto	Campo de texto	no	Puerto para la conexión con el SGBD.

3.	Usuario	Campo de texto	no	Nombre de usuario con permisos para acceder al SGBD PostgreSQL.
4.	Contraseña	Campo de texto	no	Contraseña para el usuario que accederá al SGBD PostgreSQL.
5.	Examinar	Dirección URL	no	Ventana que permite cargar el archivo mediante la ubicación especificada.

Con la utilización de un juego de datos validos e inválidos, empleando la técnica Partición de equivalencia, se evaluó y probó la validez de cada uno de los valores introducidos.

Tabla 7: Matriz de datos

Escenario	Descripción	V1	V2	V3	V4	V5	Respuesta del sistema	Flujo Central
EC1.1: Conexión con PostgreSQL	Permite la conexión con el SGBD PostgreSQL.	10.34.304.3	5432	jorge	123qwe	N/A	Lista las BD existentes en el SGBD PostgreSQL.	1. El usuario selecciona el SGBD PostgreSQL. 2. El sistema muestra las opciones de conexión. 3. Introduce los parámetros de conexión.

								4. El sistema valida los datos introducidos por el actor. 5. El sistema lista las BD existentes en el SGBD PostgreSQL. 5. El usuario selecciona la BD que desea utilizar. 6. El sistema visualiza las tablas de la BD seleccionada.
		I	V	V	V	N/A		

EC1.2: Conexión con PostgreSQL con datos inválidos.	Se verifica si todos los datos están escritos correc tamente.	Hgf.567.i	5432	jorge	123qwe		Devuelve el siguiente mensaje de error: “Error en la conexión al SGBD. Revise los parámetros de conexión”.	
		V 10.34.304.3	I efef	V jorge	V 123qwe	N/A		
		V 10.34.304.3	V 5432	I pollo	V 123qwe	N/A		
		V 10.34.304.3	V 5432	V jorge	I 123qwe	N/A		
EC1.3: Conexión con PostgreSQL con campos vacíos.	Se verifica si todos los campos han sido completado s.	I (-)	V 5432	V jorge	V 123qwe	N/A	Devuelve el siguiente mensaje de error: “Error en la conexión al SGBD. Revise los parámetros de conexión”.	
		V 10.34.304.	I (-)	V jorge	V 123qwe	N/A		
		V 10.34.304.	V 5432	I (-)	V 123qwe	N/A		
		V 10.34.304.	V 5432	V jorge	I (-)	N/A		

Escenario	Descripción	V1	V2	V3	V4	V5	Respuesta del sistema	Flujo Central
EC2.1: Conexión con SQLite.	Permite la conexión con una BD en el SGBD SQLite.	N/A	N/A	N/A	N/A	D:\TESIS\BD SQLite\BD_Syam	Conexión con una BD en SQLite.	1. El usuario selecciona la opción Examinar. 2. El sistema muestra una ventana para permitir buscar la ubicación del

								archivo de la BD SQLite. 3. El usuario selecciona el archivo de la BD deseado. 4. El usuario selecciona la opción Conectar. 5. El sistema visualiza las tablas de la BD seleccionada.
EC2.2: Conexión con SQLite errónea.	Verifica que sea cargado el archivo de la BD.	N/A	N/A	N/A	N/A	I (-)	Devuelve el siguiente mensaje de error: “Error en la conexión a la BD. Verifique el fichero seleccionado”.	

Pruebas de estrés: Con este tipo de prueba se somete al componente a niveles de carga inusuales para valorar el comportamiento del componente y el tiempo de respuesta del mismo.

Tabla 8: Análisis de los tiempos de respuesta del sistema para distintos números de tuplas.

No Prueba	Cantidad de tuplas	Tiempo de Respuesta (Milisegundos)
1	6522	35
2	7522	54
3	8522	76

4	9522	94
5	10522	115
Media	------	75

3.7.4 Resultado de las pruebas

- Con las pruebas de Caja Negra realizadas se alcanzaron resultados satisfactorios en el 90% de los casos de prueba efectuados. Los resultados que no fueron satisfactorios se convirtieron en no conformidades (NC). A continuación se presenta un ejemplo de las no conformidades encontradas en el CU Conectar a una BD.

Tabla 9: Resultados de las Pruebas

Elemento	No	No conformidad	Significativa	No significativa	Estado de NC	Resp. equipo de desarrollo
Aplicación	1	En el CU Conectar a una BD, en la sección Conexión con PostgreSQL, al introducir datos incorrectos el sistema no muestra el mensaje el error predeterminado.	X		PD 21/05/15 RA 25/05/15	Se agregó una nueva validación para mostrar mensaje de error cuando los datos introducidos sean incorrectos.
Aplicación	2	En el CU Conectar a una BD, en la sección Conexión con SQLite, al intentar cargar el fichero que contiene la BD, la aplicación no	X		PD 20/05/15 RA 26/05/15	Se agregaron los métodos correspondientes para establecer conexión con la BD.

		establece conexión con la BD.				

- Con las pruebas de Rendimiento realizadas se alcanzó una media de respuesta de 75 milisegundos para una BD a la cual se le aumentó la cantidad de tuplas desde 6522 hasta 10522.

3.8 Conclusiones parciales

La creación del diagrama de componentes sirvió para representar una vista estática de la aplicación, mostrando la organización y dependencia que existe entre los componentes físicos que se necesitan para ejecutar la misma. Los estándares de codificación y los estilos de programación utilizados fueron descritos para hacer más fácil el entendimiento del código y para permitir un mejor mantenimiento a la aplicación en un futuro. Las pruebas pertenecientes al nivel de sistema permitieron comprobar que se cumplieron los requisitos del sistema.

Conclusiones Generales

La realización del presente trabajo posibilitó cumplir con los objetivos y tareas propuestas para su desarrollo, arribándose a las siguientes conclusiones:

- El estudio de los principales conceptos relacionados al mapeo objeto-relacional permitió sentar las bases para el desarrollo del componente para el mapeo objeto-relacional en el sistema Syam.
- Con la implementación del componente para el mapeo objeto-relacional se logró el manejo de tablas dinámicas en tiempo de ejecución, además de conseguir una abstracción de la capa de acceso a datos.
- Fueron documentados los procesos de análisis, diseño, implementación y pruebas correspondientes al componente desarrollado.
- La implementación del componente para el mapeo objeto-relacional en el sistema Syam permitió dar cumplimiento a los requisitos funcionales identificados en el análisis y diseño.
- Como resultado de la investigación se obtuvo el componente para el mapeo objeto-relacional que permite el manejo de los datos, utilizando los lenguajes PostgreSQL y SQLite.
- El diseño y ejecución de las pruebas del sistema realizadas al componente, permitió validar el correcto funcionamiento del mismo, así como el cumplimiento a las funcionalidades definidas.

Recomendaciones

- Implementar para una nueva versión, las bibliotecas necesarias para el manejo de otros SGBD, como MySQL, para hacer más extensible el alcance del componente.

Referencias Bibliográficas.

PostgreSQL . 2015. The PostgreSQL Global Development Group. [En línea] 2015. [Citado el: 05 de 03 de 2015.] http://www.postgresql.org.

Alvarez, Zayas, Carlos. 1995. *METODOLOGIA DE LA INVESTIGACIÓN CIENTIFICA.* Santiago de Cuba : s.n., 1995.

Ambler, Scott W. 2005. The Agile Unified Process (AUP). *The Agile Unified Process (AUP).* [En línea] 2005. [Citado el: 25 de 05 de 2015.] http://www.ambysoft.com/unifiedprocess/agileUP.html.

Beck, K. 1999. *Extreme Programing Explained. Embrace Change.* s.l. : Pearson Education, 1999.

Booch, Grady. 1994. *The Evolution of the Booch Method.* 1994.

Booch, Grady, Rumbaugh, Jim y Jacobson, Ivar. 2006. El Lenguaje Unificado de Modelado. *UML.* 2006.

Buschmann, Regine Meunier y Hans, Rohnert. 2009. *La Guia de Arquitectura Versión 2.0.* 2009.

Cabanes, Nacho. 2007. *Introducción a las Bases de Datos.* 2007.

Camps Paré, Rafael, y otros. 2005. Bases de Datos. 2005.

CC, CODE SYNTHESIS TOOLS. 2009-2014. CODE SYNTHESIS . [En línea] 2009-2014. [Citado el: 5 de noviembre de 2014.] http://codesynthesis.com/products/odb/.

Charte, Francisco. 2006. *SQL server 2005.* s.l. : Anaya Multimedia, 2006.

Clements, Paul. 1996. *La Arquitectura de Software.* 1996.

Date, C. J. 2001. Introducción a los Sistemas de Base de Datos. México : Pearson Educación, 2001.

de la Torre, Llorente, Cesar, y otros. 2010. *Guia de arquitectura N-Capas orientada al Dominio con .NET 4.0.* España : Krasis Consulting, S.L, 2010.

Díaz Romero, Lisandra, Marín Sardinas, Yosbel y Serafín Linares, Ana Marys. 2010. *Framework de Mapeo Objeto-Relacional para PHP.* 2010.

Docsetools. 2015. Docsetools. [En línea] 2015. [Citado el: 20 de 05 de 2015.] http://docsetools.com/articulos-educativos/article_11479.html.

Doctrine. 2014. Doctrine Project. [En línea] 2014. [Citado el: 13 de 03 de 2015.] http://www.sqlitemanager.org/.

Flower, Martin. 2003. Patterns of Enterprise Application Architecture. [En línea] 01 de 2003. [Citado el: 23 de 04 de 2015.]

Gamma, Erich, y otros. 2004. *Design Patterns: Elements of Reusable Object-Oriented Software.* 2004.

Gómez Ballester, Eva, y otros. 2007. Base de Datos 1. 2007.

González Castellanos, Roberto A, Yll Lavín, Mario y Curiel Lorenzo, Lilian D. 2003. Metodología de la Investigación Científica para las Ciencias Técnicas. 2003.

Grosso, Andrés. 2011. Prácticas de Software. [En línea] 21 de 03 de 2011. [Citado el: 05 de 04 de 2015.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

Hernández, León,Rolando Alfredo y Coello, González,Sayda. *EL PROCESO DE INVESTIGACIÓN CIENTÍFICA.*

Hibernate. 2015. [En línea] JBoochDeveloper, 2015. [Citado el: 05 de 03 de 2015.] http://hibernate.org/orm/.

IBM Software Group. 2005. *Introduction to UML 2.* 2005.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El prceso unificado de desarrollo de Software.* madrid : s.n., 2000.

Jiménez, Claudia y Armstrong, Thomas. 2002. El rol del lenguaje SQL en los SGBDR y en la implementación del Modelo Relacional. [En línea] 28 de 08 de 2002. [Citado el: 05 de 05 de 2015.] http://www.inf.udec.cl/revista/edicion1/armstrong.htm.

Küng, Stefan, Onken, Lübbe y Large, Simon. 2006. *Tortoise SVN: Un cliente de subvercion para windows: Versión 1.4.1.* 2006.

Lizama Mué, Yadira y Fadraga Artiles, Lissuan. 2010. Acceso a datos en aplicaciones multihilos orientadas a objetos. Habana, Cuba : s.n., 2010, Vol. 3.

Matos, Rosa María. 1999. Diseño de Bases de Datos. 1999.

Montilva, C, Jonás A, Arapé, Nelson y Colmenares, Juan Andrés. 2008. *Desarrollo de Software Basado en Componentes.* Venezuela : s.n., 2008.

PgAdmin . 2015. PgAdmin PostgreSQL Tools. [En línea] 2015. [Citado el: 04 de 03 de 2015.] http://www.pgadmin.org.

Pressman, Roger. 2005. *Ingeniería de Software. Un enfoque práctico.* Madrid : McGraw-Hill/Interamericana de España. S.A, 2005.

QxORM, Equipo. 2011. Qt + QxOrm + Postgres. *Qt + QxOrm + Postgres.* [En línea] 18 de noviembre de 2011. [Citado el: 12 de noviembre de 2014.] http://www.qxorm.com/qxorm_en/home.html.

Rico, Jonatan Alejandro y Irvine, Zapopan. 2013. *Protocolo de proyecto para obtener el grado de Maestro en Tecnologías de la Información.* Jalisco,México : s.n., 2013.

Riggs, Simons y Krosing, Hannu. 2010. *PostgreSQL 9 Administration CooKbook.* 2010.

Rodríguez, Tamara. 2014. *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana : s.n., 2014.

Schwaber, Ken, Beedle, M y Martin, R .C. 2001. *Agile Software Development with SCRUM.* s.l. : Prentice Hall, 2001.

Silberschatz, Abraham, Korth, Henry F y Sudarshan, S. 2002. Fundamentos de Bases de Datos. 2002.

Sommerville, Ian. 2005. *Ingeniería del Software.* Madrid, España. : Pearson Addison Wesley, 2005.

SQLite Manager. 2010. SQLite Manager. [En línea] 2010. [Citado el: 20 de 03 de 2015.] http://www.sqlitemanager.org/.

SQLite. 2014. SQLite. [En línea] 2014. [Citado el: 16 de 04 de 2015.] https://www.sqlite.org/about.html.

Tadei, Leonardo. *Mapeador Objeto/Relacional: Persistencia no invasiva y por alcance en PHP.* Mar del Plata - Argentina : s.n.

The Qt Company. 2013. Qt. [En línea] 2013. [Citado el: 17 de 2 de 2015.] http://www.qt.io/qt-framework/.

Visual Paradigm. 2015. Software Desing Tools for Agile Teams, whit UML, BPMN and More. [En línea] 2015. [Citado el: 12 de 03 de 24.] http://www.visual-paradigm.com/.

Bibliografía

PostgreSQL . 2015. The PostgreSQL Global Development Group. [Online] 2015. [Cited: 03 05, 2015.] http://www.postgresql.org.

Alvarez, Zayas, Carlos. 1995. *METODOLOGIA DE LA INVESTIGACION CIENTIFICA.* Santiago de Cuba : s.n., 1995.

Ambler, Scott W. 2005. The Agile Unified Process (AUP). *The Agile Unified Process (AUP).* [Online] 2005. [Cited: 05 25, 2015.] http://www.ambysoft.com/unifiedprocess/agileUP.html.

Beck, K. 1999. *Extreme Programing Explained. Embrace Change.* s.l. : Pearson Education, 1999.

Booch, Grady. 1994. *The Evolution of the Booch Method.* 1994.

Booch, Grady, Rumbaugh, Jim and Jacobson, Ivar. 2006. El Lenguaje Unificado de Modelado. *UML.* 2006.

Buschmann, Regine Meunier and Hans, Rohnert. 2009. *La Guia de Arquitectura Versión 2.0.* 2009.

Cabanes, Nacho. 2007. *Introducción a las Bases de Datos.* 2007.

Camps Paré, Rafael, et al. 2005. Bases de Datos. 2005.

CC, CODE SYNTHESIS TOOLS. 2009-2014. CODE SYNTHESIS . [Online] 2009-2014. [Cited: noviembre 5, 2014.] http://codesynthesis.com/products/odb/.

Charte, Francisco. 2006. *SQL server 2005.* s.l. : Anaya Multimedia, 2006.

Clements, Paul. 1996. *La Arquitectura de Software.* 1996.

Date, C. J. 2001. Introducción a los Sistemas de Base de Datos. México : Pearson Educación, 2001.

de la Torre, Llorente, Cesar, et al. 2010. *Guia de arquitectura N-Capas orientada al Dominio con .NET 4.0.* España : Krasis Consulting, S.L, 2010.

Díaz Romero, Lisandra, Marín Sardinas, Yosbel and Serafín Linares, Ana Marys. 2010. *Framework de Mapeo Objeto-Relacional para PHP.* 2010.

Docsetools. 2015. Docsetools. [Online] 2015. [Cited: 05 20, 2015.] http://docsetools.com/articulos-educativos/article_11479.html.

Doctrine. 2014. Doctrine Project. [Online] 2014. [Cited: 03 13, 2015.] http://www.sqlitemanager.org/.

Flower, Martin. 2003. Patterns of Enterprise Application Architecture. [Online] 01 2003. [Cited: 04 23, 2015.]

Gamma, Erich, et al. 2004. *Design Patterns: Elements of Reusable Object-Oriented Software.* 2004.

Gómez Ballester, Eva, et al. 2007. Base de Datos 1. 2007.

González Castellanos, Roberto A, Yll Lavín, Mario and Curiel Lorenzo, Lilian D. 2003. Metodología de la Investigación Científica para las Ciencias Técnicas. 2003.

Grosso, Andrés. 2011. Prácticas de Software. [Online] 03 21, 2011. [Cited: 04 05, 2015.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

Hernández, León,Rolando Alfredo and Coello, González,Sayda. *EL PROCESO DE INVESTIGACIÓN CIENTÍFICA.*

Hibernate. 2015. [Online] JBoochDeveloper, 2015. [Cited: 03 05, 2015.] http://hibernate.org/orm/.

IBM Software Group. 2005. *Introduction to UML 2.* 2005.

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. 2000. *El prceso unificado de desarrollo de Software.* madrid : s.n., 2000.

Jiménez, Claudia and Armstrong, Thomas. 2002. El rol del lenguaje SQL en los SGBDR y en la implementación del Modelo Relacional. [Online] 08 28, 2002. [Cited: 05 05, 2015.] http://www.inf.udec.cl/revista/edicion1/armstrong.htm.

Küng, Stefan, Onken, Lübbe and Large, Simon. 2006. *Tortoise SVN: Un cliente de subverción para windows: Versión 1.4.1.* 2006.

Lizama Mué, Yadira and Fadraga Artiles, Lissuan. 2010. Acceso a datos en aplicaciones multihilos orientadas a objetos. Habana, Cuba : s.n., 2010, Vol. 3.

Matos, Rosa María. 1999. Diseño de Bases de Datos. 1999.

Montilva, C, Jonás A, Arapé, Nelson and Colmenares, Juan Andrés. 2008. *Desarrollo de Software Basado en Componentes.* Venezuela : s.n., 2008.

PgAdmin . 2015. PgAdmin PostgreSQL Tools. [Online] 2015. [Cited: 03 04, 2015.] http://www.pgadmin.org.

Pressman, Roger. 2005. *Ingeniería de Software. Un enfoque práctico.* Madrid : McGraw-Hill/Interamericana de España. S.A, 2005.

QxORM, Equipo. 2011. Qt + QxOrm + Postgres. *Qt + QxOrm + Postgres.* [Online] noviembre 18, 2011. [Cited: noviembre 12, 2014.] http://www.qxorm.com/qxorm_en/home.html.

Rico, Jonatan Alejandro and Irvine, Zapopan. 2013. *Protocolo de proyecto para obtener el grado de Maestro en Tecnologías de la Información.* Jalisco,México : s.n., 2013.

Riggs, Simons and Krosing, Hannu. 2010. *PostgreSQL 9 Administration CooKbook.* 2010.

Rodríguez, Tamara. 2014. *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana : s.n., 2014.

Schwaber, Ken, Beedle, M and Martin, R .C. 2001. *Agile Software Development with SCRUM.* s.l. : Prentice Hall, 2001.

Silberschatz, Abraham, Korth, Henry F and Sudarshan, S. 2002. Fundamentos de Bases de Datos. 2002.

Sommerville, Ian. 2005. *Ingeniería del Software.* Madrid, España. : Pearson Addison Wesley, 2005.

SQLite Manager. 2010. SQLite Manager. [Online] 2010. [Cited: 03 20, 2015.] http://www.sqlitemanager.org/.

SQLite. 2014. SQLite. [Online] 2014. [Cited: 04 16, 2015.] https://www.sqlite.org/about.html.

Tadei, Leonardo. *Mapeador Objeto/Relacional: Persistencia no invasiva y por alcance en PHP.* Mar del Plata - Argentina : s.n.

The Qt Company. 2013. Qt. [Online] 2013. [Cited: 2 17, 2015.] http://www.qt.io/qt-framework/.

Visual Paradigm. 2015. Software Desing Tools for Agile Teams, whit UML, BPMN and More. [Online] 2015. [Cited: 03 12, 24.] http://www.visual-paradigm.com/.

Glosario de Términos

Actor: Es algo con comportamiento, ya sea una persona (identificada por un rol), un sistema informatizado u organización, que realiza algún tipo de interacción con el sistema.

Automatizar: Se le denomina así a cualquier tarea realizada por máquinas en lugar de personas. Es la sustitución de procedimientos manuales por sistemas de cómputo.

Caso de Uso (CU): Es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso.

Diagrama de Casos de Uso (DCU): Muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa.

Framework: Esquema, esqueleto o patrón para el desarrollo y/o la implementación de una aplicación.

Mineral: Sustancia natural que se diferencia del resto por su origen inorgánico, su homogeneidad, composición química preestablecida y que corrientemente ostenta una estructura de cristal.

Multi-plataforma: Sistema informático que corre sobre varios sistemas operativos, sin prescindir de ninguna de sus funcionalidades.

Requerimiento: Petición de una acción que se considera necesaria.

Yacimiento Mineral: Es la acumulación natural de minerales en la corteza terrestre, en forma de uno o varios cuerpos minerales, los cuales en este estado, pueden ser objeto de extracción y explotación industriales, en la actualidad o en un futuro inmediato.