Universidad de las Ciencias Informáticas

FACULTAD 6

Título: Componente para el acceso a datos con soporte multihilo en Qt.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

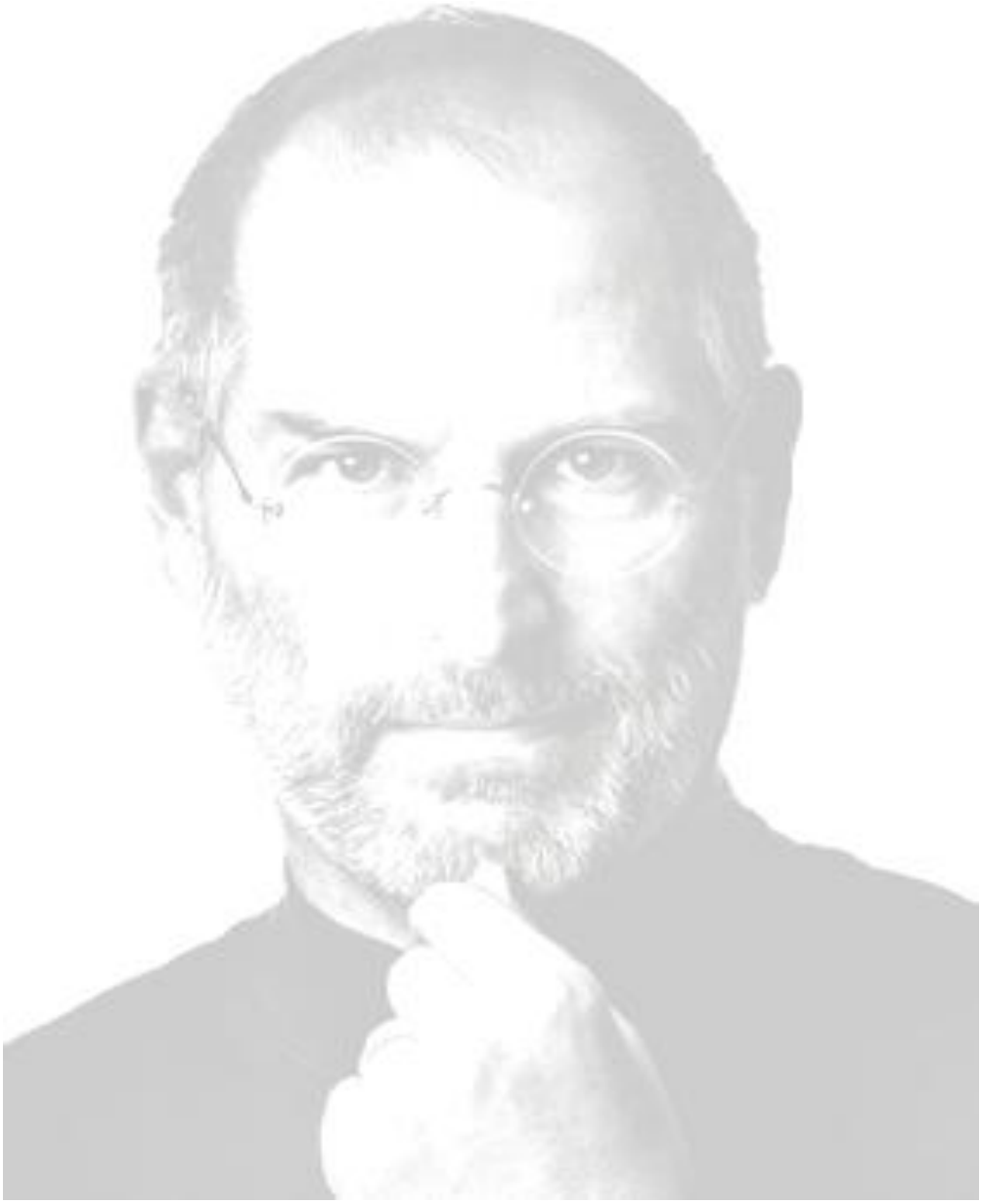
Autor(es): Cheila Aguila Alvarado

Roxanna Aliaga Noguera

Tutor: Ing. Cesar Santos Sanabria

Co-Tutor: Ing. Dorgis Montes de Oca Gómez

Junio de 2013



…la innovación es lo que distingue a un líder de los demás…

Steve Jobs

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Cheila Aguila Alvarado

_____

Firma del Autor

Roxanna Aliaga Noguera

_____

Firma del Autor

Cesar Santos Sanabria

_____

Firma del Tutor

Dorgis Montes de Oca Gómez

_____

Firma del Co-Tutor

Datos de Contacto

Tutor: Ing. Cesar Santos Sanabria
Categoría: Especialista General
Centro de trabajo: Universidad de las Ciencias Informáticas
Título de la especialidad de graduado: Ingeniero en ciencias Informáticas
Año de graduación: 2010
Institución en la que se graduó: Universidad de las Ciencias Informáticas.
Correo electrónico: csanabria@uci.cu

Co- Tutor: Ing. Dorgis Montes de Oca Gómez
Categoría: Adiestrado
Centro de trabajo: Universidad de las Ciencias Informáticas
Título de la especialidad de graduado: Ingeniero en ciencias Informáticas
Año de graduación: 2012
Institución en la que se graduó: Universidad de las Ciencias Informáticas.
Correo electrónico: dorgis@uci.cu

Dedicatoria

Roxanna:

A mi mama Migdalia y a mi papá José por haberme dado la vida primero que todo y haber estado ahí apoyándome y cuidándome en todo momento.

A mi hermano Leandro espero servirte como guía, para que seas un universitario también.

A mi tía Caridad por ser mi segunda mamá.

A toda mi familia por ser esas bellas personas que siempre han estado ahí para mí.

Cheila

Dedico especialmente este trabajo a mi madre que ha sido mi soporte y guía…

A mis abuelos que siempre me han apoyado…

A mi novio que siempre ha estado ahí cuando más lo he necesitado…

A mi familia por el apoyo incondicional…

A mi primo Mandy y a mi hermano Giry, espero darles razones suficientes para que sean ellos quienes puedan dedicar su tesis…

A mis Amigos que son la familia que se puede escoger…

Agradecimientos

Roxanna:

A mi mamá Migdalia por ser esa mujer que ha estado ahí para cuidarme y ayudarme en todos los momentos de mi vida, por haberme dado las razones suficientes para que fuera universitaria y ser esa madre que todos deseamos tener. Te quiero mucho y estoy muy orgullosa de ti.

A mi papá José por tener ese carácter fuerte que ha hecho que valore y piense mejor a la hora de tomar cualquier decisión en mi vida, por ser ese padre que piensa en su familia por sobre todas las cosas. Gracias papi por estar ahí para mí siempre.

A mi tía por ayudarme y quererme tanto. Gracias por enseñarme que la familia es lo primero.

A mi hermano por ser esa personita especial que está a mi lado siempre. Te quiero mucho.

A mi novio por haberme hecho tan feliz estos meses y mostrarme un lado de la vida que no conocía.

A mis amigas del alma Yanet y Yicel (las mellizas) por haber conseguido que cambiara en cierta medida mi modo de pensar y haber influido junto con mi mamá para que esté aquí hoy. Gracias a las dos por haberme soportado estos 8 años, las quiero mucho.

A mi amiga Silvia y su ya casi esposo Osmeli, gracias por quererme aún con mis cambios de ánimo.

A mis amigos "los inteligentes" Asiel, Yunier y Humberto, por ayudarme a pasar por las pruebas y los proyectos y por ser mis amigos antes que todo.

A mi compañera de tesis por soportarme a lo largo de este curso, sé que no fue fácil miji, pero muchas gracias.

Agradecimientos

A mi tutor y mi co-tutor, por apoyarme y ayudarme con la tesis en especial a Dorgis que nunca estaba conforme y siempre nos pidió el 100%.

A mis compañeros de aula, a todos, en especial a los de primer año.

A los que no he mencionado por olvidadiza, pero me han apoyado en mi vida y en mi carrera.

Cheila:

El paso por la universidad me ha permitido crecer como persona, como mujer y como profesional, este sueño no podría haberse realizado sin la guía y el amor incondicional de mi Madre a quien le debo ser hoy la persona que soy, mami aunque no te lo diga mucho eres la persona más importante de mi vida y Te Quiero Muchísimo…

Agradezco a mis Abuelos Lucia y Giraldo por saber que siempre estarán ahí para lo que necesite por su cariño y su confianza.

A mi bisabuela Basilia que sé, que este donde este, me ha estado cuidando todo este tiempo, te recuerdo mucho….

A mi novio, amigo y futuro esposo Dorgis por guiarme, ayudarme, quererme y estar siempre conmigo al mismo lado del camino.

A mis tíos y a mi papá especialmente a mi tío Javier por su cariño y preocupación en todo momento.

A Jorge por su preocupación, su apoyo, su cariño pero sobre todo por hacerme saber que mi mamá está en las mejores manos.

A mis amigos de siempre Idanelis, Sandy y Ode, por demostrarme que no importa cuánto tiempo pase siempre estarán ahí para mí.

Agradecimientos

A mis amigos del IPVCE Luismy y Carballo por ser siempre incondicionales…

A mis amigas Yudith, Daysi y Diana quienes han vivido conmigo tantos buenos y malos momentos en estos 5 años, quienes sin importar hora han escuchado mis problemas y los han solucionado muchas veces también, el destino puede que separe nuestros caminos pero siempre serán uno de mis mejores recuerdos de la universidad.

A los muchachos del grupo Yanes (por escuchar siempre mis problemas), Ale (mi pinareño preferido), Hectico (un ejemplo de solidaridad y compañerismo), Daryel, Darian, Suao, Alejandro y todos en general…

A mi compañera de tesis por haber compartido conmigo estos meses de trabajo, y tantos momentos de nerviosismo.

A mi tutor Cesar y todos los profesores que de una forma u otra me han formado como Ingeniera en Ciencias Informáticas...

Resumen

La producción de software en Cuba ha ampliado sus horizontes con la aparición de las aplicaciones de código abierto o software libre. La Universidad de las Ciencias Informáticas (UCI) no está exenta a estos cambios, por lo que los proyectos productivos han tenido que comenzar a migrar parte de sus productos. En el caso específico del Centro de Geoinformática y Señales Digitales (GEYSED), para muchas de sus aplicaciones *desktop*[1] se está usando el *framework* Qt.

Por los requisitos de estos proyectos, se debe acceder a las base de datos ya sea para almacenar, modificar o consultar datos. Cuando se realiza el acceso concurrentemente a los datos desde el *framework* Qt el implementador debe cerrar y abrir conexiones por cada consulta que exista en cada hilo, esta acción es propensa a la aparición de errores humanos ya que en ocasiones la conexión de un hilo podría no ser cerrada por el implementador, lo que incurriría en errores en el código. Además, cada proyecto accede de forma distinta a la base de datos incurriendo en errores tales como: código repetitivo, malas prácticas de programación y las aplicaciones estén atadas a un Sistema Gestor de Base de Datos en específico.

Para atenuar estos problemas se desarrolla el componente para el acceso a datos con soporte multihilo que es capaz de crear varios hilos y elevar la calidad de la capa de acceso a datos, además de que permite el mapeo relacional de objetos sin necesidad de usar las tradicionales consultas SQL[2].

Palabras Claves: *framework* Qt, multihilo, software libre.

[1] Desktop: Aplicación de escritorio.

[2] SQL: Lenguaje de Consulta Estructurado

Índice

TABLA DE FIGURAS

Introducción

Las Tecnologías de la Información y las Comunicaciones, conocidas por las siglas TIC, han evolucionado a una velocidad vertiginosa y representan un ícono en el desarrollo de cualquier sociedad. Cuba no ha quedado rezagada, y como parte de sus proyectos por insertarse en el desarrollo del mundo de la informatización, ha creado instituciones afines con el tema y ha intensificado los programas en las carreras de ciencias informáticas y cibernética. Una de estas instituciones es la Universidad de las Ciencias Informáticas, que surge en el año 2002 por idea del Comandante en Jefe Fidel Castro Ruz.

Desde sus inicios, la misión de la UCI ha estado enfocada en formar profesionales altamente calificados en la rama de la informática y sobre todo, comprometidos con la Patria y la Revolución. Además, producir aplicaciones y servicios informáticos que sirvan de soporte a la industria cubana de la informática, a partir de la vinculación estudio-trabajo como modelo de formación.

La producción de software en Cuba ha ampliado sus horizontes con la aparición de las aplicaciones de código abierto o software libre. Su aplicación garantiza al país soberanía tecnológica, evitando la pérdida de autonomía a la que conlleva el uso de programas de código propietario. Es por esto que el país tiene como estrategia la migración hacia el software libre, lo que proporcionaría una mayor seguridad en términos de soporte informático. Con la construcción de aplicaciones informáticas de código abierto, las grandes empresas privadas de la industria del software perderían fuerza e irían quedando a la zaga.

La UCI no está exenta a estos cambios, por lo que los proyectos productivos han tenido que comenzar a migrar parte de sus productos a software libre. En el caso específico del Centro de Geoinformática y Señales Digitales (GEYSED), para muchas de sus aplicaciones *desktop* se está usando el *framework* Qt.

Por los requisitos de estos proyectos, se debe acceder a las base de datos constantemente ya sea para almacenar, modificar o consultar datos. Cuando se realiza el acceso concurrentemente a los datos desde el *framework* Qt el implementador debe cerrar y abrir conexiones por cada consulta que exista en cada hilo, esta acción es propensa a la aparición de errores humanos ya que en ocasiones la conexión de un hilo podría no ser cerrada por el implementador, lo que incurriría en errores en el código. Además, cada proyecto accede de forma distinta a la base de datos incurriendo en errores tales como: código repetitivo, malas prácticas de programación (objetos de la capa de presentación en la capa de acceso a datos, mal manejo de memoria, alto acoplamiento, baja cohesión, propenso a inyección SQL, no se usa ningún

patrón de acceso a datos), las aplicaciones están atadas a un Sistema Gestor de Base de Datos en específico.

El entendimiento de la **problemática** anterior ha desencadenado esfuerzos en aras de mejorar el acceso a datos en el departamento, surgiendo de esta forma el siguiente problema: ¿Cómo elevar la calidad de la capa de acceso a datos en los productos desarrollados con el *framework* Qt y lenguaje de programación C++ en el departamento Señales Digitales?

A partir de la situación planteada, se define como **objeto de estudio** el mapeo relacional de objetos con soporte multihilo como técnica de acceso a datos y como **campo de acción** el acceso a datos en los proyectos que conforman el departamento Señales Digitales

Siendo el **objetivo general** del trabajo: Desarrollar un componente para el acceso a datos con soporte multihilo con el *framework* Qt.

Para alcanzar el objetivo trazado se debe dar cumplimiento a los siguientes objetivos específicos:

- Crear un componente que permita el mapeo de objetos desde la programación orientada a objetos a las bases de datos que utilizan el modelo relacional.
- Diseñar soporte para acceso concurrente a la capa de acceso a datos y gestión de memoria automática.
- Ejecutar pruebas de caja negra para detectar el correcto funcionamiento de los requisitos funcionales del componente para acceso a datos.

Para guiar el desarrollo de este proceso se plantea la siguiente **idea a defender:**

Si se desarrolla un componente para el acceso a datos con soporte multihilo y con el *framework* Qt, se elevará la calidad de la capa de acceso a datos en los productos del departamento Señales Digitales.

Tareas de la investigación:

- Caracterización de los procesos relacionados con el acceso a datos con soporte multihilo.
- Caracterización de los componentes para acceso a datos con soporte multihilo.
- Caracterización del proceso de mapeo con relaciones simples al modelo relacional.

- Caracterización del proceso para soporte para acceso a datos de forma concurrente.
- Caracterización del proceso de gestión de memoria automática.
- Caracterización del *framework* y el lenguaje seleccionado para desarrollar el componente para acceso a datos con soporte multihilo.
- Selección de la metodología de desarrollo de software a utilizar.
- Definición de la arquitectura del componente para acceso a datos con soporte multihilo.
- Caracterización de los patrones de acceso a datos.
- Identificación de los requisitos funcionales y no funcionales del componente para acceso a datos con soporte multihilo.
- Selección la herramienta CASE[3] a utilizar para el modelado del componente para acceso a datos con soporte multihilo.
- Diseño de las clases del sistema.
- Implementación de las clases del sistema.
- Selección de la prueba a ejecutar al componente para acceso a datos con soporte multihilo.
- Diseño de la prueba a ejecutar al componente para acceso a datos con soporte multihilo.
- Aplicación de pruebas al componente para acceso a datos con soporte multihilo.

Para obtener los resultados esperados se usaron algunos **métodos científicos** que forman parte del proceso de la investigación. Los métodos teóricos utilizados fueron:

[3] CASE: Ingeniería de Software Asistida por Computadora.

Histórico – Lógico

Se basa en el estudio histórico del Mapeo Objeto-Relacional (ORM por sus siglas en inglés), analizando la trayectoria completa de los mismos y determinando cuáles han sido las etapas fundamentales de su expansión. Ponen de manifiesto la lógica interna en el desarrollo del mapeo objeto-relacional y permite unir el estudio de la estructura de los ORM con su concepción histórica.

Analítico – Sintético

Para el estudio del Mapeo Objeto Relacional se indaga en varias bibliografías, dividiéndolas por elementos facilitando así su estudio. Luego se establece la unión entre todas las partes previamente analizadas, permitiendo formar las características generales y las principales relaciones entre ellas.

Capítulo I: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se muestra la fundamentación teórica del actual trabajo. Se expondrán los principales conceptos asociados al tema para una mejor comprensión de los lectores. Además, se analizará el objeto de estudio planteado para luego exponer el estado del arte analizando las soluciones existentes. También se abordarán las tendencias y tecnologías utilizadas y se especificará la metodología de desarrollo a utilizar.

1.2 Principales conceptos asociados

1.2.1. El Mapeo Objeto-Relacional (ORM)

El Mapeo Objeto Relacional es una técnica de programación para relacionar datos entre un lenguaje de programación orientado a objetos y una base de datos relacional. Uno de los modelos de bases de datos usados en la actualidad es el relacional. El modelo de objetos difiere en muchos aspectos del modelo relacional. La herramienta que une esos dos modelos se llama mapeo relacional-objeto (Object Relational Mapping, 2008).

El uso de los ORM posibilita la utilización de características propias de la orientación a objetos, facilitando el trabajo de los implementadores y logrando una interrelación entre el modelo relacional y el orientado a objetos. Esta técnica permite eliminar, insertar y actualizar datos persistentes en una base de datos relacional.

1.2.2. Base de Datos (BD)

Las bases de datos son una colección de archivos interrelacionados, que son creados con un DBMS[4]. El contenido de una base de datos engloba a la información concerniente (almacenadas en archivos) de una organización, de tal manera que los datos estén disponibles para los usuarios, una finalidad de la base de datos es eliminar la redundancia o al menos minimizarla. Los tres componentes principales de un sistema de base de datos son el hardware, el software DBMS y los datos a manejar, así como el personal encargado del manejo del sistema (Espinoza, 2012).

[4] DBMS: Sistemas Administradores de Bases de Datos

Las bases de datos surgen como almacenes en los que se permite guardar un gran conjunto de información perteneciente a un tema específico. La misma estará estructurada y organizada de manera que la forma en que están almacenados los datos sea entendible por sus usuarios.

1.2.3. Bases de Datos Relacional (BDR)

Las BDR son aquellas que su modelo de datos es el relacional y los datos se muestran en forma de tablas y relaciones. Las tablas se representan gráficamente como una estructura rectangular formada por filas y columnas. Cada columna almacena información sobre una propiedad determinada de la tabla (se le llama también atributo), nombre, dni, apellidos, edad. Cada fila posee una ocurrencia o ejemplar de la instancia o relación representada por la tabla (a las filas se las llama también tuplas). (Sánchez, 2004)
Con las bases de datos relacionales se pueden establecer relaciones entre los datos de forma sencilla y planificada, son usadas en la actualidad para los tradicionales sofware de gestión. Las mismas cuentan con la deficiencia de que permiten una pobre representación de aspectos del mundo real.

1.2.4. Bases de Datos Orientadas a Objeto (BDOO)

Las BDOO son aquellas cuyo modelo de datos está orientado a objetos y almacenan y recuperan objetos en los que se almacena estado y comportamiento. Su origen se debe a que en los modelos clásicos de datos existen problemas para representar cierta información, puesto que aunque permiten representar gran cantidad de datos, las operaciones que se pueden realizar con ellos son simples. Las clases utilizadas en un determinado lenguaje de programación orientado a objetos son las mismas clases que serán utilizadas en una BDOO; de tal manera, que no es necesaria una transformación del modelo de objetos para ser utilizado por un Sistema de Gestión de Base de Datos Orientada a Objetos (SGBDOO). De forma contraria, el modelo relacional requiere abstraerse lo suficiente como para adaptar los objetos del mundo real a tablas (Alberca, y otros, 2011) .

Las BDOO surgen para atenuar los problemas existenten en las bases de datos relacionales, ya que estas presentan problemas al representar aspectos del mundo real, a pesar de las ventajas que las mismas poseen no se ha logrado que se puedan realizar con ellas diversidad de operaciones los que las torna limitadas.

1.2.5. Bases de Datos Objeto Relacional (BDOR)

Las BDOR son bases de datos que desde el modelo relacional evolucionan hacia bases de datos más extensas y complejas, incorporando para obtener este fin, conceptos del modelo orientado a objetos. Podemos decir que un Sistema de Gestión de Bases de Datos Objeto-Relacional (SGBDOR) contiene dos tecnologías; la tecnología relacional y la tecnología de objetos. En una base de datos objeto-relacional se siguen almacenando tuplas, aunque la estructura de las tuplas no está restringida a contener escalares (tipos compuestos como vectores, conjuntos, etc.) sino que las relaciones pueden ser definidas en función de otras, que se denomina herencia directa (Alberca, y otros, 2011).

Las BDOR integran ambos modelos con la intención de lograr una base de datos más completa que permita relacionar tanto aspectos del modelo orientado a objetos, como del modelo relacional. Las mimas permiten gestionar tipos de datos complejos y pueden ir adaptando las aplicaciones y bases de datos para que utilicen las funciones orientadas a objetos.

1.2.6. Sistema Gestor de Base de Datos (SGBD)

Un Sistema Gestor de Base de Datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos. Ayuda a realizar las siguientes acciones: (Pérez, y otros, 2012)

- Definición de los datos.
- Mantenimiento de la integridad de los datos dentro de la base de datos.
- Control de la seguridad y privacidad de los datos.

1.2.7. Sistema de Gestión de Base de Datos Orientada a Objetos (SGBDOO)

Los Sistemas de Gestión de Bases de Datos Orientadas a Objetos surgen debido a la falta de capacidad semántica del modelo relacional para atender nuevos tipos de aplicaciones: (Pérez, y otros, 2012)

- Bases de datos gráficas y de imágenes.

- Bases de datos científicas.
- Sistemas de información geográfica.
- Bases de datos de multimedia.
- Acceso uniforme a sistemas de múltiples bases de datos.

Este tipo de aplicaciones exige trabajar con datos de forma diferente a los conocidos porque necesitan: (Pérez, y otros, 2012)

- Estructuras complejas para los objetos.
- Transacciones de mayor duración.
- Nuevos tipos de datos para almacenar imágenes o grandes bloques de texto.
- Necesidad de definir operaciones no estandarizadas, específicas para cada aplicación.
- Controlar versiones y configuraciones.

1.2.8. Lenguaje de Consulta Estructurado (SQL)

El Lenguaje de Consulta Estructurado(SQL), por sus siglas en inglés (*Structured Query Language*), es un lenguaje estándar de definición y manipulación (y consulta) de bases de datos relacionales. El SQL estándar incluye: (Computación, 2012)

- Características del álgebra relacional.
- Características del cálculo relacional de tuplas.

SQL es un estándar en el lenguaje de acceso a bases de datos. Originalmente, era un lenguaje de acceso al sistema de gestión de bases de datos denominado DB2 en plataformas 390 de IBM[5]. (Mastermagazine, 2012)

Las principales funcionalidades de SQL como Lenguaje de Definición de Datos (DDL) son la creación, modificación y borrado de las tablas que componen la base de datos, así como de los índices, vistas, sinónimos, permisos, entre otros que pudieran definirse sobre las mismas. (Cursos, 2012)

1.2.9. Programación Orientada a Objetos (POO)

La Programación Orientada a Objetos (POO) constituye una forma especial de programar, donde se enfoca el problema a la vida real, esto la distingue de las demás formas de programación.

La POO es una manera de diseñar y desarrollar software que trata de imitar la realidad tomando algunos conceptos esenciales de ella; el primero es precisamente el de objeto, cuyos rasgos son la identidad, el estado y el comportamiento: (Cursodejava, 2012)

- La identidad es el nombre que distingue a un objeto de otro.
- El estado son las características que lo describen.
- El comportamiento es lo que puede hacer.

1.2.10. Framework

El concepto *framework* se emplea un muchos ámbitos del desarrollo de sistemas software, no solo en el ámbito de aplicaciones web. Podemos encontrar *frameworks* para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, y para cualquier otro ámbito. (Gutiérrez, 2012)

En general, el término *framework se refiere* a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un *framework* se puede considerar como una aplicación genérica incompleta y configurable a la que se le puede añadir las últimas piezas para construir una aplicación concreta. (Gutiérrez, 2012)

[5] IBM: *International Business Machines*, empresa internacional estadounidense.

Los objetivos principales que persigue un *framework* son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. (Gutiérrez, 2012)

1.3 Análisis del objeto de estudio

En la actualidad debido al acelerado desarrollo de las TIC, la mayoría de los bancos de datos se encuentran en formato electrónico. Consecuentemente varias organizaciones se encargan de encontrar mejores soluciones al problema de almacenamiento de datos. Los Sistemas Gestores de Bases de Datos permiten que se pueda almacenar y acceder a la información de forma rápida y estructurada. Es notable su uso en disímiles organizaciones y entornos ya que son de vital importancia en esferas científicas y productivas.

En la literatura mundial existen varios modelos de bases de datos que son abstracciones que van a permitir que un sistema de bases de datos sea implementado de manera eficaz. Algunos frecuentes son las Bases de Datos Orientadas a Objetos, las Bases de Datos Relacionales y las Bases de Datos Objeto-Relacionales.

Las Bases de Datos Orientadas a Objetos atenúan problemas tales como modelar ciertos aspectos del "mundo real", pues los modelos frecuentes permiten representar gran cantidad de datos, pero con la deficiencia que solo se pueden efectuar operaciones simples. Además de las dificultades que genera el paso del modelo de objetos al modelo relacional. En conclusión, surgen para tratar de mitigar las deficiencias de los modelos anteriores y para proporcionar eficiencia y sencillez a las aplicaciones.

Las principales características asociadas a las BDOO son: (Alberca, y otros, 2011)

- **Objetos**: Cada entidad del mundo real se modela como un objeto. La forma de identificarlos es mediante un identificador de objetos (OID, *Object Identifier*), único para cada uno de ellos. Generalmente este identificador no es accesible ni modificable para el usuario (modo de aumentar la integridad de entidades y la integridad referencial). Los OID son independientes del contenido. Es decir, si un objeto cambia los valores de atributos, sigue siendo el mismo objeto con el mismo OID. Si dos objetos tienen el mismo estado pero diferentes OID, son equivalentes pero tienen identidades diferentes.

- **Encapsulamiento**: Cada objeto contiene y define procedimientos (métodos) y la interfaz mediante la cual se puede acceder a él y otros objetos pueden manipularlo. La mayoría de los SGBDOO permite el acceso directo a los atributos, incluyendo operaciones definidas por el propio SGBDOO, las cuales leen y modifican los atributos para evitar que el usuario tenga que implementar una cantidad considerable de métodos cuyo único propósito sea el de leer y escribir los atributos de un objeto. Generalmente, los SGBDOO permiten al usuario especificar qué atributos y métodos son visibles en la interfaz del objeto y pueden invocarse desde afuera.

Las desventajas de un SGBDOO son: (Alberca, y otros, 2011)

- Carencia de un modelo de datos universal. No hay ningún modelo de datos que esté universalmente aceptado para los SGBDOO y la mayoría de los modelos carecen una base teórica.

- Carencia de experiencia. Todavía no se dispone del nivel de experiencia del que se dispone para los sistemas relacionales.

- Carencia de estándares. Existe una carencia de estándares general para los SGBDOO.

- Competencia. Con respecto a los SGBDR y los SGBDOR, estos productos tienen una experiencia de uso considerable. SQL es un estándar aprobado y ODBC[6] es un estándar de facto. Además, el modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales.

- La optimización de consultas compromete la encapsulación, requiere una compresión de la implementación de los objetos, para poder acceder a la base de datos de manera eficiente.

- El modelo de objetos aún no tiene una teoría matemática coherente que le sirva de base.

[6] ODBC: *Open DataBase Connectivity*, estándar de acceso a las bases de datos.

Principales características de las Bases de Datos Objeto-Relacionales: (Alberca, y otros, 2011)

- Con las Bases de Datos Objeto-Relacional, se pueden crear nuevos tipos de datos, que permiten gestionar aplicaciones complejas con una gran riqueza de dominios. Estos pueden ser tipos compuestos, lo que implica que se debe definir al menos dos métodos transformadores, uno para convertir el tipo nuevo a ASCII[7] y otro que convierte de ASCII al nuevo tipo.
- Se soportan tipos complejos como registros, conjuntos, referencias, listas, pilas, colas y arreglos.
- Se pueden crear funciones que tengan un código en algún lenguaje de programación como por ejemplo: SQL, Java, C, etc.
- Existe una mayor capacidad expresiva para los conceptos y asociaciones.
- Se pueden crear operadores asignándole un nombre y existencia de nuevas consultas con mayor capacidad consultiva.
- Se soporta el encadenamiento dinámico y herencia en los tipos tuplas o registro.
- Se pueden compartir varias bibliotecas de clases ya existentes, esto es lo que conocemos como reusabilidad.
- Posibilidad de incluir el chequeo de las reglas de integridad referencial a través de los *triggers.*
- Soporte adicional para seguridad y activación de la versión cliente-servidor.

Principales características de las Bases de Datos Relacionales: (Bases de Datos Relacional, 2011)

- Una Base de Datos Relacional se compone de varias tablas o relaciones.
- No pueden existir dos tablas con el mismo nombre ni registro. Cada tabla es a su vez un conjunto de registros (filas y columnas).

[7] ASCII: Código Estándar Estadounidense para el Intercambio de Información.

- La relación entre una tabla padre y un hijo se lleva a cabo por medio de las claves primarias y ajenas (o foráneas).
- Las claves primarias son la clave principal de un registro dentro de una tabla y éstas deben cumplir con la integridad de datos.
- Las claves foráneas se colocan en la tabla hija, contienen el mismo valor que la clave primaria del registro padre; por medio de éstas se hacen las relaciones.

Las debilidades y limitaciones de los SGBDR son: (Alberca, y otros, 2011)

- Pobre representación de las entidades del 'mundo real'.
- Sobrecarga y poca riqueza semánticas.
- Soporte inadecuado para las restricciones de integridad y empresariales.
- Estructura de datos homogénea.
- Operaciones limitadas.
- Dificultades para gestionar las consultas recursivas.
- Desadaptación de impedancias.
- No ofrecen soporte para tipos definidos por el usuario (sólo dominios).

En la actualidad las bases de datos relacionales continúan siendo utilizadas para las tradicionales aplicaciones de software de gestión. Por otra parte la Programación Orientada a Objetos ha creado un nuevo paradigma en la programación gracias a las ventajas que esta ofrece en cuanto a facilidades para los implementadores, se conoce que entre el modelo de datos orientado a objetos (el utilizado en la capa de aplicaciones) y el relacional (utilizado en la capa de almacenamiento de los datos) existe una incoherencia llamada Diferencia por Impedancia Objeto-Relacional, en inglés *O/R Impedance Mismatch*.

Esto corresponde a que el paradigma orientado a objetos se basa en los principios de la ingeniería y el paradigma relacional está fundamentado en principios matemáticos.

A continuación un pequeño ejemplo gráfico de la diferencia por impedancia:

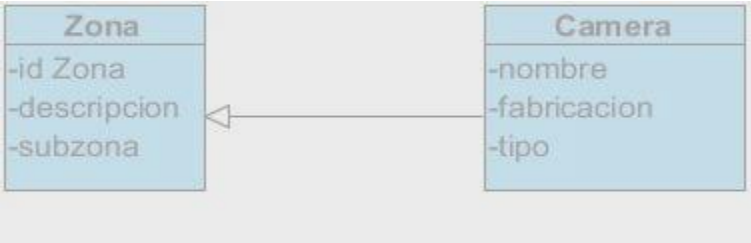


Figura 1: Diagrama de clases Zona-Camera.

Una instancia de la clase Camera quedaría de la siguiente forma:

Figura 2: Instancia de la clase Camera.

Al crear una instancia de la clase camera se pueden acceder a los atributos heredados de la clase zona. Es decir, desde un objeto de tipo **Camera** se accede a su propiedad **nombre, fabricación y tipo**, también a las heredadas de la clase **Zona** (**id, descripción y subzona**). A continuación el diagrama entidad-relación correspondiente a este modelo. (Romero, y otros, 2010)

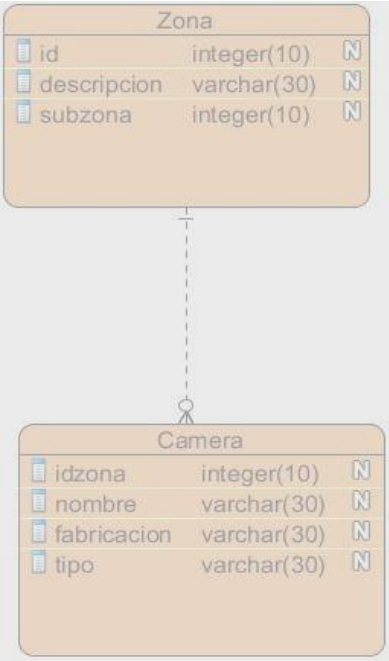


Figura 3: Diagrama entidad relación Zona-Camera.

A simple vista se ve la imposibilidad de representar herencia en un modelo relacional, para poder seleccionar una camera completa se realizaría a través de una sentencia SQL parecida a la que se muestra a continuación: (Romero, y otros, 2010)

SELECT * **FROM** *Zona*, *Camera* **WHERE** *Camera.Id_Zona = Zona.Id* **AND** *otros criterios de selección*.

La entidad **Camera** en el modelo relacional ofrece solamente los atributos **nombre, fabricación, tipo** y una referencia a la entidad relacionada **Zona**. Otro problema es, si se restringe el acceso a la clase **Zona** declarándola como abstracta, solamente se pueden crear objetos de tipo **Camera**, lo que quiere decir que las únicas instancias que existen de la clase **Zona** son las propias instancias de la clase **Camera**, en cambio en el modelo relacional no se tienen opciones para esto, es decir se puede tener un registro de la tabla **Zona** que no esté relacionado con ningún registro de la tabla **Camera**. (Romero, y otros, 2010)

Como se ha ilustrado en el ejemplo anterior, no es posible representar un objeto y sus relaciones tal y como son en el modelo relacional. De ahí que el surgimiento de los ORM brinde la solución a la Diferencia por Impedancia y facilite el manejo de datos relacionales desde la capa de aplicación usando POO.

Entre los aspectos que caracterizan el Mapeo Objeto-relacional se encuentran los siguientes: (Object Relational Mapping, 2008)

- Técnica de programación para relacionar datos entre un lenguaje de programación orientado a objetos y una base de datos relacional.
- Una estrategia de persistencia de objetos en una base de datos.
- Una herramienta para comunicar el mundo de la POO con una base de datos relacional.
- Una herramienta para soportar la independencia de los datos y los lenguajes de programación orientados a objetos.

Como ventajas del Mapeo Objeto-Relacional se encuentran: (Object Relational Mapping, 2008)

- Independencia de la lógica con el SGBD.
- Reusabilidad.
- Madurez.
- Disminución del tiempo de desarrollo.
- Disminución del costo de desarrollo.
- Diseño, implementación y mantenimiento de la arquitectura.

La reusabilidad es una ventaja que aportan los ORM que permite llamar a los métodos de un objeto de datos desde diferentes lugares de una aplicación e incluso desde aplicaciones diferentes. Esta capa encapsula también la lógica de los datos, de esta manera cuando se necesite solo se invoca un método

del modelo de datos y cuando se necesite realizar algún cambio en el funcionamiento del método no se tiene que modificar la aplicación y solo se modifica el método de cálculo.

Existe otra consideración importante que hay que tener en cuenta cuando se crean elementos de acceso a los datos: las empresas que crean las bases de datos utilizan variantes diferentes del lenguaje SQL. Si se cambia a otro sistema gestor de bases de datos, es necesario reescribir parte de las consultas SQL que se definieron para el sistema anterior. Si se crean las consultas mediante una sintaxis independiente de la base de datos y un componente externo se encarga de traducirlas al lenguaje SQL concreto de la base de datos, se puede cambiar fácilmente de una base de datos a otra. Este es precisamente el objetivo de las capas de abstracción de bases de datos. Esta capa obliga a utilizar una sintaxis específica para las consultas y a cambio realiza el trabajo de optimizar y adaptar el lenguaje SQL a la base de datos concreta que se está utilizando. (Zaninotto, y otros, 2008)

La portabilidad constituye una ventaja importante porque hace posible el cambiar la aplicación a otra base de datos en cualquier momento durante el desarrollo del proyecto. Si se quiere desarrollar rápidamente un prototipo de una aplicación y el cliente no ha decidido todavía la base de datos que mejor se ajusta a sus necesidades, se puede construir la aplicación con una base de datos de apoyo y cuando el cliente haya tomado la decisión, cambiar fácilmente a los sistemas gestores de bases de datos MySQL, PostgreSQL u Oracle. Cambiando solamente una línea en un archivo de configuración todo funcionará correctamente.

A continuación se muestra en un ejemplo gráfico, la ubicación del Mapeo Objeto –Relacional en un sistema que comprende desde la capa de vista hasta el DBMS:

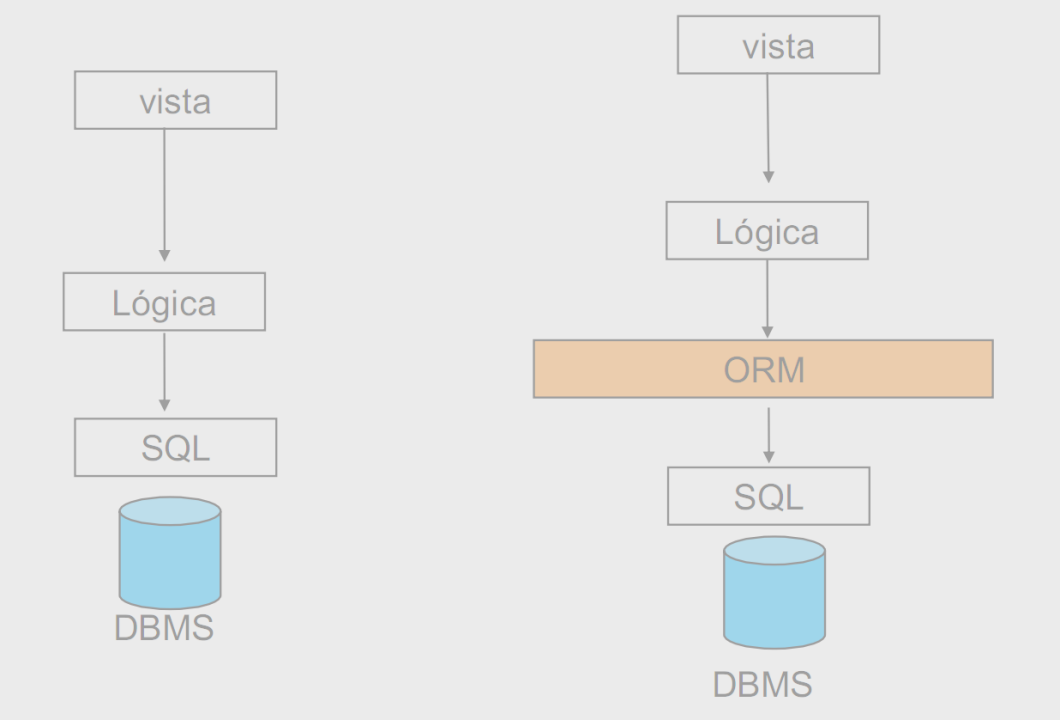


Figura 4: Ubicación del Mapeo Objeto –Relacional.

Se puede definir que un ORM permite: (Mapeo del modelo de objetos al modelo relacional , 2005)

- Mapear clases a tablas: propiedad a columna, clase a tabla.
- Persistir objetos. A través de un método Orm.Save (objeto). Encargándose de generar el SQL correspondiente.
- Recuperar objetos persistidos.
- Recuperar una lista de objetos a partir de un lenguaje de consulta especial.

En la problemática antes analizada se evidencia la necesidad del implementador de cerrar las conexiones de los hilos utilizados para las consultas a la base de datos, aquí es donde el ORM viene a suplir esta necesidad, ya que el mismo se encarga de realizar esta tarea de forma transparente al desarrollador, creando un mapeo de bases de datos con soporte multihilo. Cada proceso usando la programación multihilo empieza utilizando un único hilo, pero derivado de cualquiera de sus hilos puede establecer hilos adicionales.

Dentro de un proceso puede haber varios hilos de ejecución (varios *threads*). Eso quiere decir que un proceso podría estar haciendo varias cosas "a la vez". Los hilos dentro de un proceso comparten todos, la misma memoria. Si un hilo modifica una variable, todos los demás hilos del mismo proceso verán el nuevo valor de la variable, lo que hace necesario el uso de semáforos. (Ruiz, 2009).

Para aplicaciones que interactúan con bases de datos, la concurrencia o uso simultáneo de varias circunstancias se analiza desde dos contextos: el acceso concurrente a través de múltiples conexiones de usuarios al mismo conjunto de datos y la manipulación concurrente de una misma conexión por múltiples hilos de ejecución de la aplicación.

1.4 Análisis de soluciones existentes

1.4.1. Qdjango, ODB y QxORM

El ORM QDjango es un *framework* web escrito en C++ y construido sobre la biblioteca de Qt. Siempre que sea posible trata de seguir a Django[8], de ahí su nombre. Se distribuye bajo los términos de la Licencia Pública General Reducida (LGPL) versión 2.1 o posterior. (QDjango, 2012)

Entre las características que posee se encuentran: (QDjango, 2012)

- Soporte para una amplia gama de base de datos.
- Creación y registro de los modelos de base de datos es fácil usando Qt.
- Realización de consultas.
- Puede crear y eliminar tablas de bases de datos e índices para los modelos registrados.

ODB es un sistema de código abierto, multiplataforma de mapeo objeto-relacional para C++. Permite la persistencia de objetos en una base de datos relacional sin tener que lidiar con las tablas, columnas o SQL manualmente y sin escribir ningún código de asignación. Está licenciado mediante Licencia Pública General (GPL) versión 3.

[8] Django: Es un *framework* de desarrollo web de código abierto, escrito en Python.

Tiene como características: (Synthesis, 2012)

- Permitir la generación automática de códigos para la base de datos de apoyo.
- Capacidad para manejar estándares de códigos en C++.
- Permitir la generación automática del esquema de base de datos.
- Permitir la generación de un esquema de base de datos independiente.
- Apoyo a la ejecución de sentencias SQL.

En el caso de QxORM es un ORM multiplataforma que proporciona un excelente soporte y documentación. Usando Qt se puede escribir un simple y eficaz código C++, es compatible con una gran cantidad de objetos de Qt: QObject, QString, QDate, QTime, QDateTime, Qlist, QHash, QSharedPointer, QScopedPointer. (QxORM, 2012)

A pesar de las ventajas que ofrecen estos software para el mapeo objeto-relacional no se ha podido ejecutar su uso, ya que la licencia bajo la que están liberados los ORM, GPL y LGPL, no permite que sean usados por el departamento Señales Digitales, ya que obliga al mismo a distribuir el código de fuente de las aplicaciones, y a liberar los software realizados bajo esta misma licencia dificultando la comercialización de los mismos. Además está el hecho de que se dificultaría darle mantenimiento a los ORM en el caso que existiera la necesidad de agregarle alguna funcionalidad. Pero el estudio de los mismos ha guiado la confección del componente para acceso a datos en cuestión.

1.5 Metodologías y herramientas utilizadas el desarrollo de la solución

1.5.1. Metodología de desarrollo.

La metodología de software certifica la calidad del software, el mismo debe estar sujeto a normas de calidad pues el énfasis en el proceso de desarrollo asegura un producto adecuado a los requisitos de los clientes. Las metodologías de software son una guía para los desarrolladores, pero la gran diversidad de aplicaciones que existe en la actualidad deriva una gran variedad de metodologías.

Una metodología conocida y usada por los desarrolladores es RUP (*Rational Unified Process*, por sus siglas en inglés) que es una metodología flexible ya que puede modificarse y ajustarse a las necesidades del proyecto. Es una metodología pesada pero también podría usarse de una manera ágil, solo tiene que ser adaptada al ambiente en el que se desarrolla. Por ello, por la amplia bibliografía con la que cuenta esta metodología y debido a que es la que actualmente se usa en el proyecto Video-Vigilancia se escoge como guía para la confección del componente para el acceso a datos.

RUP puede utilizar el Lenguaje de Modelado Unificado (UML) para la elaboración de todos los diagramas de un sistema de software y los sustentan tres principios: dirigido por casos de uso, centrado en la arquitectura e iterativo, e incremental.

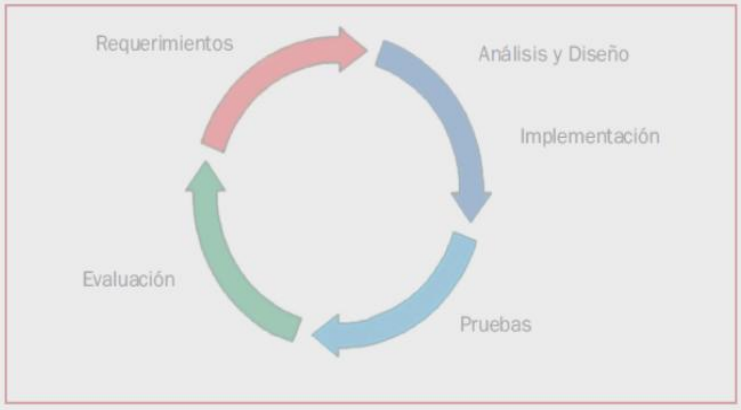


Figura 5: Etapas del desarrollo del software.

Determina las etapas a realizar durante el proyecto de creación del software. (Isaías Carrillo Pérez, y otros, 2008)

- Ingeniería o modelado del negocio: Analizar y entender las necesidades del negocio para el cual se está desarrollando el software.
- Requisitos: Proveer una base para estimar los costos y tiempo de desarrollo del sistema.
- Análisis y diseño: Trasladar los requisitos analizados anteriormente a un sistema automatizado y desarrollar una arquitectura para el sistema.
- Implementación: Crear un software que se ajuste a la arquitectura diseñada y que tenga el comportamiento deseado.

- Pruebas: Asegurarse de que el comportamiento requerido es correcto y que todo lo solicitado está presente.
- Despliegue: Producir distribuciones del producto y distribuirlo a los usuarios.

Figura 6: Diagrama del esfuerzo de actividades según las etapas del proyecto.

La metodología se encarga no solo de definir las etapas de desarrollo sino de garantizar la calidad del producto final, estas características la hacen óptima para la confección del componente para acceso a datos con soporte multihilo.

1.5.2. Lenguaje Unificado de Modelado versión 2.0.

El Lenguaje Unificado de Modelado (UML: *Unified Modeling Language*) se ha convertido en ese estándar ansiado para representar y modelar la información con la que se trabaja en las fases de análisis y especialmente de diseño. El lenguaje UML tiene una notación gráfica expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de

uso, el diseño con los diagramas de clases, objetos, etc., hasta la implementación y configuración con los diagramas de despliegue. (Orallo, 2002)

UML es además un método formal de modelado, aporta las siguientes ventajas: (Orallo, 2002)

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de alto nivel, de la estructura de un proyecto.

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones: (Orallo, 2002)

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: a partir de los modelos se pueden construir los sistemas diseñados.
- Documentar: los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Para una mejor comprensión se hace necesaria una representación gráfica la que favorece la visión del análisis del problema. Se selecciona UML para la construcción de los diagramas que ofrecen la vista del sistema a modelar y por tanto facilita la comprensión a los desarrolladores del problema a resolver.

1.5.3. Herramienta CASE

Para aumentar la productividad en el desarrollo de software, disminuir los costos en tiempo y dinero y mejorar la planificación de un proyecto hay disímiles herramientas destinadas a estas funciones. Las mismas están presentes en el ciclo de desarrollo de un software sobre todo en la realización del diseño

del proyecto. Es indispensable prestarle atención a la selección del tipo de herramienta CASE a utilizar, ya que existe una gran variedad: Microsoft Project, Racional Rose, JDeveloper, Visual Paradigm, entre otras.

Visual Paradigm versión 8.0 es una herramienta CASE que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar y compatible entre ediciones. (Alonso, 2009)

Características principales: (Alonso, 2009)

- Permite realizar ingeniería inversa a un proyecto: código a modelo, código a diagrama.
- Distribución automática de diagramas.

La utilización de esta herramienta para la modelación del producto está justificada en que no se integra al producto final por lo que no tendría trascendencia legal, además de las ventajas antes mencionadas. Es una herramienta multiplataforma, es decir que, para la migración que se propone el país no supondría ningún problema.

1.5.4. Lenguaje de programación C++

El lenguaje de programación **C++** es un lenguaje imperativo orientado a objetos derivado del **C**. En realidad un súper conjunto de **C**, que nació para añadirle cualidades y características de las que carecía. El resultado es que como su ancestro, sigue 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. (Zator Systems, 2012)

C++ se trata simplemente del sucesor de un lenguaje de programación hecho por programadores (de alto nivel) para programadores, lo que se traduce en un diseño pragmático al que se le han ido añadiendo todos los elementos que la práctica aconsejaba como necesarios. (Zator Systems, 2012)

Es un lenguaje de programación basado en C que soporta directamente conceptos de la orientación a objetos. Desde el punto de vista de la POO ofrece recursos como son las formas de encapsulamiento, la herencia y el polimorfismo. Los mayores beneficios se aprecian a largo plazo: alta calidad del software, mayor reutilización de código y mayor facilidad de adaptación, esta última apreciada sobre todo en grandes proyectos.

La utilización de este lenguaje le proporcionará al software la robustez necesaria para crear un producto de alta calidad. Se logrará una mayor reutilización del código con el empleo de este flexible lenguaje de programación.

1.5.5. *Framework* de desarrollo Qt

El *framework* de desarrollo Qt es un marco de trabajo multiplataforma para el desarrollo de aplicaciones con interfaz gráfica de usuario o de consola. Presenta un gran número de características entre las que se destacan las siguientes:

- Es portable desde sistemas de escritorio hasta sistemas operativos embebidos.
- Herramientas y entornos de desarrollo multiplataforma.
- Alto rendimiento en todas las plataformas.

Cuenta con abundante documentación, arquitectura que soporta el uso de plugins y hasta el momento ha sido liberado bajo dos licencias, la LGPL y la comercial (techerald.com, 2010).

El uso de este *framework* garantiza una compatibilidad entre el componente para acceso a datos y las aplicaciones implementadas en el departamento Señales Digitales además de una estructura al software que contribuirá a una buena organización y rápido desarrollo. Posee una gran cantidad de clases y funcionalidades en C++ que ayudarán de manera decisiva a construir un producto fiable.

1.6 Conclusiones.

Luego de realizar el estudio de los fundamentos teóricos de la investigación y contando con una propuesta de solución al problema planteado, se concluye que:

- El estudio de las soluciones existentes en la actualidad permitió conocer que no existe un componente para el mapeo objeto-relacional que satisfaga las necesidades actuales del departamento Señales Digitales.
- El desarrollo de un componente para acceso a datos con soporte multihilo constituye una solución factible al problema a resolver.
- La selección de RUP como metodología de desarrollo garantiza una guía de construcción a lo largo de todas las etapas del ciclo de vida del software, así como la documentación compatible con la existente en el proyecto.
- Las herramientas y tecnologías seleccionadas para la construcción del sistema propuesto son multiplataforma lo que garantiza que cumplan con la política de migración a software libre de la UCI.

Capítulo II Presentación de la Solución Propuesta

2.1 Introducción

En el presente capítulo se realiza una descripción de la solución propuesta. El objetivo fundamental del mismo es dar a conocer los requisitos funcionales y no funcionales del sistema, realizándose una descripción detallada de cada uno para un mejor entendimiento de la solución que se propone. Para el desarrollo del presente capítulo se mantendrán los elementos definidos en la fundamentación teórica anteriormente expuestos.

2.2 Modelo del dominio

La diversidad de necesidades de los implementadores en el momento de acceder a la base de datos provoca que este proceso se desarrolle de forma variable. Cada proceso puede acceder al recurso en el momento que lo necesite, es por ello que no existe un proceso de negocio visible y bien definido para estas funcionalidades. Lo que trae consigo que no sea posible realizar el modelo de negocio quedando como solución la elaboración del modelo de dominio.

El modelo del dominio se utiliza con frecuencia como fuente de inspiración para el diseño de los objetos de software. Es un artefacto importante que se crea durante el análisis orientado a objetos. Un modelo del dominio es una representación de las clases conceptuales del mundo real, no de componentes software. No se trata de un conjunto de diagramas que describen clases del software, u objetos del software con responsabilidades. Es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. (Larman, y otros, 2003)

2.2.1 Diagrama de clases del modelo del dominio

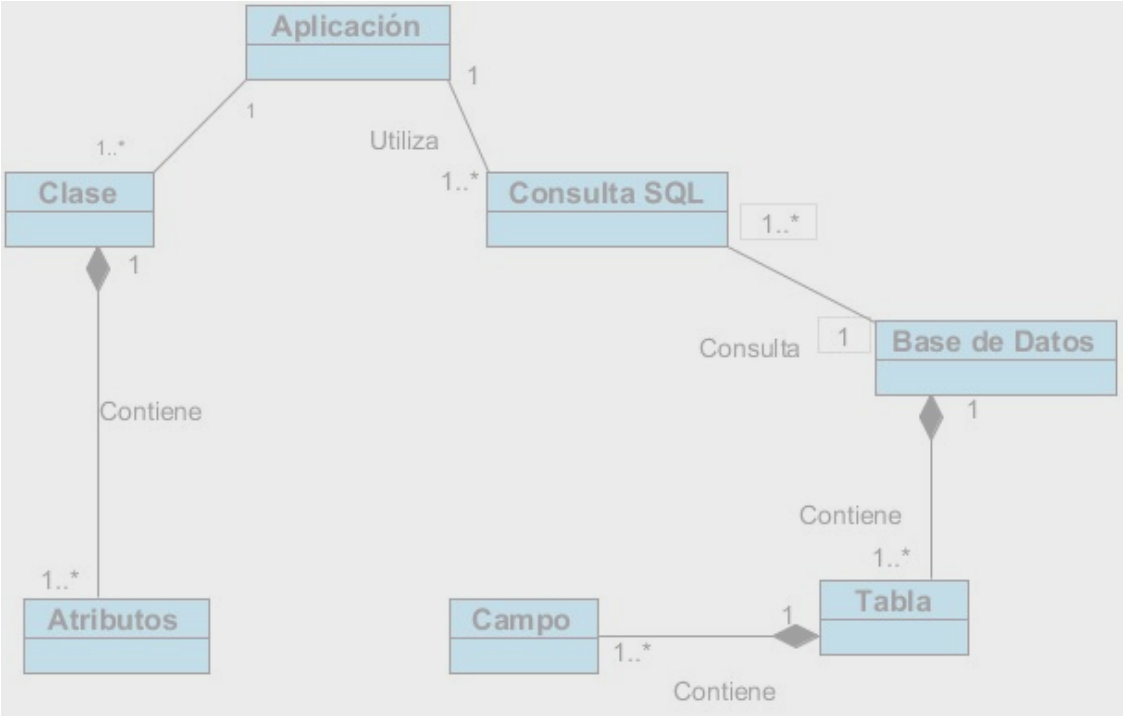


Figura 7: Diagrama de clases del modelo de dominio.

El diagrama representa una aplicación que estará compuesta por clases que a su vez están compuestas por atributos, la misma realiza una o muchas consultas SQL a la base de datos cuando necesite acceder a la información de la misma, la base de datos está compuesta una o muchas tablas y cada tabla por uno o muchos campos.

2.3 Requisitos de software

Los requisitos están sustentados a partir de la necesidad de perfeccionar y agilizar el trabajo de los implementadores en el departamento Señales Digitales.

2.3.1. Requisitos funcionales

Los requisitos funcionales (RF) son la definición de los servicios que el sistema debe proporcionar, cómo debe reaccionar a una entrada particular y cómo se debe comportar ante situaciones particulares. (Laguna, 2011)

Estos requisitos describen las funciones del sistema con sus entradas, salidas y excepciones, dependen del tipo de software desarrollado y de los posibles usuarios que este tendría.

- RF1 Realizar la actualización de objetos clases persistentes. La aplicación debe poder modificar la información contenida en las tuplas de las tablas que forman parte de la base de datos relacional.

- RF2 Realizar la inserción de objetos clases persistentes. El componente debe añadir la información especificada en las tuplas que forman parte de la base de datos relacional.

- RF3 Realizar la eliminación de objetos clases persistentes. La aplicación debe eliminar información de las tuplas que formen parte de las tablas de la base de datos según sea necesario.

- RF4 Consultar los objetos contenidos en la base de datos dado un criterio. Buscar el objeto que este contenido en los registros de las tablas de la base de datos relacional dado un criterio.

- RF5 Materializar los datos del modelo relacional a objetos de clases persistentes. El componente debe ser capaz de mapear las tuplas que forman parte de las tablas de la base de datos relacional y convertirlas a objetos clases persistentes.

- RF6 Desmaterializar objetos de clases persistentes al modelo relacional. El componente debe ser capaz de convertir objetos clases persistentes a tuplas de las tablas del modelo relacional.

- RF7 Acceder a los objetos contenidos en la BD de forma transparente desde diferentes hilos. El componente debe permitir crear varios hilos y que se puedan hacer traslados de conexiones entre hilos, es decir, tener soporte para concurrencia.

2.3.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) son restricciones que afectan a los servicios o funciones del sistema, tales como restricciones de tiempo sobre el proceso de desarrollo, estándares entre otras. (Laguna, 2011)

Las características que hacen del componente un producto atractivo, rápido y confiable serán los requisitos no funcionales que se tendrán en cuenta en el momento de confeccionar el software. A continuación se realiza la descripción de los requisitos que se tendrán en cuenta en el presente trabajo.

2.3.2.1 Compatibilidad

Lograr la compatibilidad entre el componente para acceso a datos y las aplicaciones desarrolladas en Qt del departamento Señales Digitales. Para ello se implementa el componente para acceso a datos usando el *framework* Qt y el lenguaje de programación C++.

2.3.2.3 Usabilidad

La utilización del sistema debe ser permitida a implementadores que trabajen con el *framework* Qt y lenguaje de programación C++. Para ello se crea un manual de usuario en el que se encuentra explicado cómo se realiza la interacción con el componente.

2.3.2.4 Rendimiento

Agilizar la materialización de objetos de la base de datos. Para ello cuando se mapea la base de datos relacional se guarda la información temporalmente en la memoria de la PC por lo que se evita tener que establecer una nueva conexión a la base de datos, mejorando en tiempo las respuestas.

2.3.2.5 Requisito de Hardware

El uso del componente para acceso a datos requiere de un hardware que permita soportar el trabajo con el mismo. Para que el sistema funcione correctamente se recomiendan como requisitos de hardware:

- Memoria RAM de 1Gb o superior.

- Disco Duro de 80 Gb o superior.
- Procesador Intel Core 2 Duo de 2 GHz o superior.

2.4 Modelo de casos de uso del sistema

1.2.11. Definición de los actores del sistema

Un actor especifica un rol que adopta una entidad externa (usuario, hardware externo u otro sistema) que interacciona directamente con el sistema.(Modelado Básico con Casos de Uso)
Se define como actor del sistema a toda **aplicación externa** que necesite facilitar el proceso de desarrollo de aplicaciones en el departamento Señales Digitales que utilice información contenida en una base de datos relacional.

1.2.12. Diagrama de casos de uso del sistema

Los **diagramas de casos de uso** documentan el comportamiento de un sistema desde el punto de vista del usuario. Por lo tanto, los casos de uso determinan los requisitos funcionales del sistema, es decir, representan las funciones que un sistema puede ejecutar. (Diagramas de Casos de Uso)

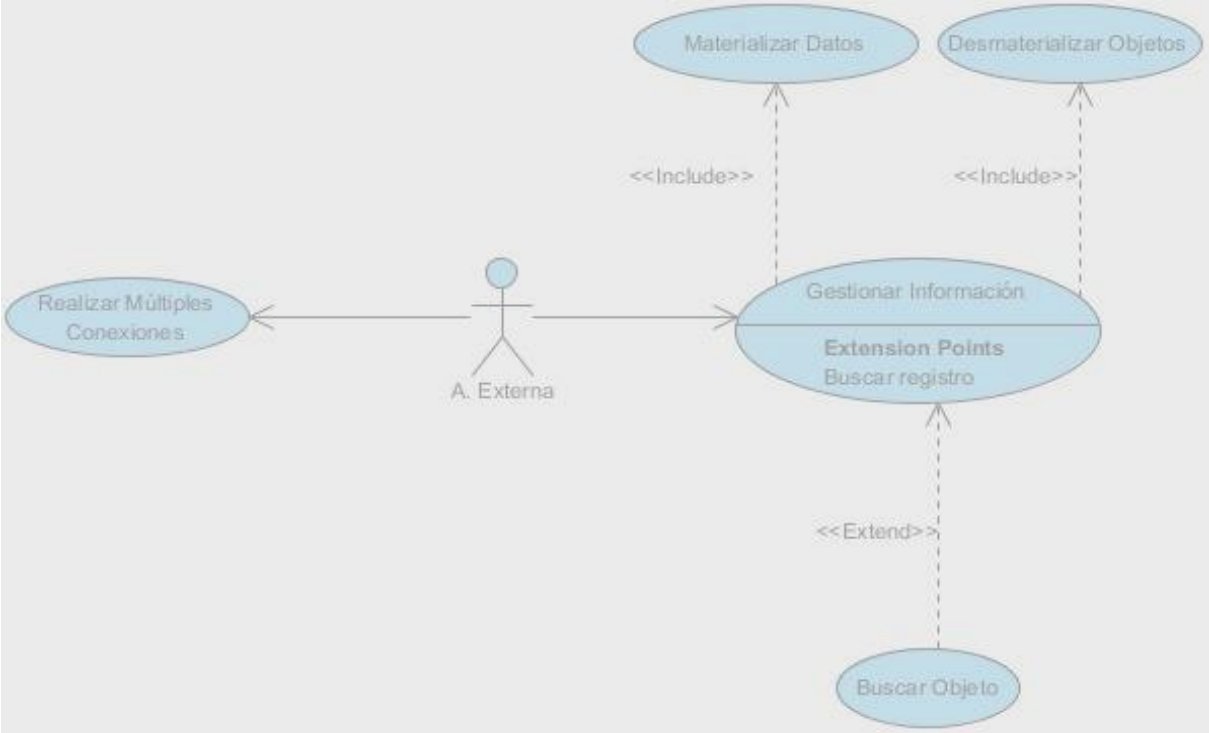


Figura 8: Diagrama de casos de uso del sistema

2.5 Descripción textual de los casos de uso del sistema

Caso de Uso:	Gestionar Información.
Propósito:	Manipular la información de la BD.
Actores:	Aplicación externa.
Resumen:	El Caso de Uso se inicia cuando el desarrollador necesita insertar, actualizar o eliminar información de la base de datos. Para ello define los parámetros de conexión a la misma. Concluye cuando el desarrollador actualiza, inserta o elimina información de la base de datos.
Precondiciones:	La base de datos se encuentre registrada.

Poscondiciones:	Se actualizan los registros de las tablas.
Referencias:	RF 1, RF 2, RF3.
Prioridad	Crítica.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
1. El actor define los parámetros de conexión a la base de datos.	2. El sistema comprueba que los datos de conexión sean correctos.
3. El actor realiza una de las siguientes acciones.	4. Si realiza: a. Insertar información, ver sección “Insertar información”. b. Actualizar información, ver sección “Actualizar información”. c. Eliminar información, ver sección “Eliminar información”. d. Buscar información. **Ir al caso de uso Buscar objeto.**
Sección Insertar Información	
1. El actor inserta la información deseada.	2. El sistema desmaterializa los objetos**. Ir al caso de uso Desmaterializar Objetos.** 3. El sistema añade la información especificada en las tuplas que forman parte de la base de datos relacional y la base de datos queda actualizada con el elemento insertado terminando así el caso de uso.

Sección Actualizar Información	
1. El actor busca la información que desea modificar. 3. El actor modifica la información deseada.	2. El sistema busca la información. **Ir al caso de uso Buscar Objeto.** 4. El sistema actualiza la información especificada en las tuplas que forman parte de la base de datos relacional terminando así el caso de uso.
Sección Eliminar Información	
1. El actor busca la información que desea modificar. 3. El actor elimina la información deseada.	2. El sistema busca la información. **Ir al caso de uso Buscar Objeto.** 4. El sistema elimina la información especificada en las tuplas que forman parte de la base de datos relacional terminando así el caso de uso.

Caso de Uso:	Buscar Objeto.
Propósito:	Manipular la información de la BD.
Actores:	Aplicación externa.
Resumen:	El Caso de Uso se inicia cuando la aplicación necesita buscar un objeto de la base de datos. Termina cuando logra encontrar o no el objeto deseado en la base de datos y lo muestra al usuario en caso de

	ser encontrado en la base de datos.
Precondiciones:	La base de datos se encuentre registrada.
Poscondiciones:	Se consultan los registros de las tablas.
Referencias:	RF 4.
Prioridad	Crítica.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
1. El actor escribe los parámetros de la búsqueda.	2. El sistema desmaterializa los objetos. **Ir al caso de uso Desmaterializar Objetos.** 3. El sistema busca en la base de datos los parámetros especificados. 4. El sistema materializa los datos de las tuplas de la base de datos. **Ir al caso de uso Materializar Datos.** 4. El sistema de devuelve los datos buscados terminando así el caso de uso.
Flujo Alterno de Eventos	
	1. El sistema no encuentra el elemento buscado y lanza un error informando que no ha sido encontrado el elemento en la base de datos.

Caso de Uso:	Materializar Datos.
Propósito:	Obtener la información en el modelo orientado a objetos.
Actores:	Aplicación externa.
Resumen:	El Caso de Uso se inicia cuando el actor necesita convertir los parámetros del modelo relacional al orientado a objetos.
Precondiciones:	La base de datos se encuentre registrada.
Poscondiciones:	Se materializan los datos de la base de datos.
Referencias:	RF5.
Prioridad	Crítica.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
	2. El sistema mapea las tuplas que forman parte de las tablas de la base de datos relacional y las convierte a objetos clases persistentes terminando así el caso de uso.

Caso de Uso:	Desmaterializar Objetos.
Propósito:	Obtener la información en el modelo orientado a objetos.
Actores:	Aplicación externa.
Resumen:	El Caso de Uso se inicia cuando el actor necesita convertir los parámetros del modelo orientado a objetos al relacional.
Precondiciones:	La base de datos se encuentre registrada.

Poscondiciones:	Se desmaterializan los datos de la base de datos.
Referencias:	RF6.
Prioridad	Crítica.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
	1. El sistema convierte objetos de clases persistentes a tuplas de las tablas del modelo relacional terminando así el caso de uso.

Caso de Uso:	**Múltiples conexiones**
Propósito:	Realizar varias conexiones a la base de datos
Actores:	Aplicación externa.
Resumen:	El Caso de Uso se inicia cuando la aplicación necesita realizar varias conexiones al mismo tiempo.
Precondiciones:	La base de datos se encuentre registrada.
Poscondiciones:	Se realicen varios métodos indicados por el usuario concurrentemente.
Referencias:	RF 7.
Prioridad	Alta.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor define los métodos que desea	2. El sistema permite realizar múltiples

realizar concurrentemente.	conexiones a la base de datos y realiza concurrentemente los métodos deseados terminando así el caso de uso.

2.6 Conclusiones

Una vez generados los artefactos que constituyen una guía previa a la implementación del sistema en cuestión, se concluye que:

- Luego de estudiar los conceptos fundamentales relacionados con el contexto del sistema propuesto, se decidió elaborar el Modelo del dominio, teniendo en cuenta que los procesos del negocio no se encuentran bien definidos.
- Durante la etapa de análisis fueron identificados siete requisitos funcionales, los cuales fueron agrupados en cinco CUS. Se definieron, además, los requisitos no funcionales que responden a las necesidades de hardware y software para el funcionamiento adecuado del sistema.
- Se obtuvo una representación visual del actor con los casos de uso que inicializa mediante el desarrollo del diagrama de casos de uso, estableciendo un punto de partida para el posterior diseño del componente propuesto.

Capítulo III Construcción de la Solución Propuesta

3.1 Introducción.

Este capítulo se basa en la confección de la solución propuesta, se muestra un modelado de las clases que se han implementado para lograr una visualización de los elementos a realizar. Los diagramas de clases del diseño elaborados a continuación describen la estructura y comportamiento que tendrá el componente para acceso a datos con soporte multihilo una vez concluida la etapa de implementación. Se detallan además todos los patrones de diseño que se han utilizado para lograr un correcto funcionamiento de la aplicación.

3.2 Arquitectura basada en componentes.

La complejidad de los sistemas computacionales actuales ha llevado a buscar la reutilización del software existente. El desarrollo de software basado en componentes permite reutilizar piezas de código pre-elaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión. (Terreros, 2013)

Esta arquietectura divide el sofware en componentes funcionales los cuales se mejorarán con el tiempo, ya que cada implementador puede optimizar los componentes de forma individual antes de ver el sistema integrado. Se escoge esta arquitectura ya que un componente para acceso a datos debe ser perfeccionado según los nuevos requisitos que necesite una aplicación o un implementador. Cuando se usa una arquitectura basada en componentes gracias al bajo acoplamiento de los mismos las funcionalidades pueden ser implementadas por desarrolladores diferentes pues sería sencillo integrar solo un componente individual.

Beneficios del Desarrollo de Software Basado en Componentes

En esencia, un componente es una pieza de código pre-elaborado que encapsula alguna funcionalidad expuesta a través de una interfaz estándar. Los componentes son los "ingredientes de las aplicaciones", que se juntan y combinan para llevar a cabo una tarea. Es algo similar a lo que se puede observar en un equipo de música. Cada componente del mismo ha sido diseñado para acoplarse perfectamente con sus pares, las conexiones son estándar y el protocolo de comunicación está ya preestablecido. Al unirse las partes, se obtiene música. (Terreros, 2013)

El paradigma de ensamblar componentes y escribir código para hacer que estos componentes funcionen se conoce como Desarrollo de Software Basado en Componentes. El uso de este paradigma posee algunas ventajas: (Terreros, 2013)

1. **Reutilización del software.** Lleva a alcanzar un mayor nivel de reutilización de software.
2. **Simplifica las pruebas.** Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
3. **Simplifica el mantenimiento del sistema.** Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
4. **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.

Figura 9: Representación de los componentes del ORM implementado

El componente para el acceso a datos continuaría su correcto funcionamiento sin la presencia de los componentes que la forman, estos son:

- Componente Multihilo. Se encarga de realizar varias consultas a la base de datos de simultáneamente.
- Componente FindbyID. Se encarga de buscar un elemento en la base de datos dado un identificador.
- Componente FindAll. Se encarga de buscar todos los objetos de la base de datos.

3.3 Patrones de diseño

Los patrones de diseño de software constituyen una solución reusable y comprobada a problemas que son comunes en el desarrollo de software.

Un patrón es una solución que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en el desarrollo de sistemas software. No son bibliotecas de clases, sino un "esqueleto" básico que cada desarrollador adapta a las peculiaridades de su aplicación. Los patrones se describen en forma textual, acompañados de diagramas (habitualmente de clases e interacción) y pseudocódigo. Se deben distinguir de los estilos arquitectónicos y, en particular, de los marcos de trabajo. (Pimentel, 2007)

3.3.1 Patrón Instancia Única

El patrón Instancia única o *Singleton* en inglés, garantiza que una clase solamente tenga una instancia y proporciona un punto de acceso global a la instancia. (León, 2005)

Este patrón se implementa en la clase SingletonBase y con ayuda de los semáforos controla el acceso a las zonas críticas.

3.3.2 Patrón Mapa Identidad

El patrón Mapa Identidad o *Identity Map* en inglés, evita tener en memoria dos representaciones distintas del mismo objeto en una transacción de negocio; funciona como memoria caché de objetos del negocio; como corolario, minimiza las lecturas a la base de datos. (Paredes, 2008)

Se implementa en la clase AbstractMapper y su principal funcionalidad es darle seguimiento a un objeto evitando que sea mapeado más de una vez en la misma transacción de negocio.

3.3.3 Patrón Registro

Registro o *Registry* en inglés, es un medio simple y eficiente de compartir datos y objetos en una aplicación sin tener que preocuparse de mantener numerosos parámetros o hacer uso de variables globales. (Màrius, 2007)

Se implementa en la calse MappersRegistry en el momento que se registra la base de datos.

3.3.4 Patrón *Data Mapper*

Se define como una capa de mapeo que mueve datos entre objetos y una base de datos mientras se mantienen independientes uno de otro del mapeo en sí mismo. *Data Mapper* es una capa de software que separa los objetos cargados en memoria de la base de datos. Su responsabilidad es transferir datos entre la base de datos y los objetos, también es el encargado de aislar uno del otro. Con el *Data Mapper* los objetos no necesitan conocer la base de datos ni las interfaces del código SQL. (Martin Fowler , 2001)

Se implenta en la clase AbstractMapper y en cada clase mapper que se crea para cada tabla de la base de datos.

3.4 Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en los requisitos funcionales como en los no funcionales. Las abstracciones del modelo de diseño tienen una correspondencia directa con los elementos físicos del ambiente de implementación. (Torossi, 2008)

Con los diagramas de clases se representan las clases que son utilizadas en el sistema para lograr visualizar las relaciones existentes entre ellas tales como herencia, composición, agregación y asociación. Permite la fácil interpretación por los programadores descubriendo fallas del sistema en el diseño. A continuación se muestran los diagramas de clases del diseño correspondientes a cada caso de uso del sistema.

3.4.1 Diagrama de clases del diseño para el caso de uso Materializar Datos

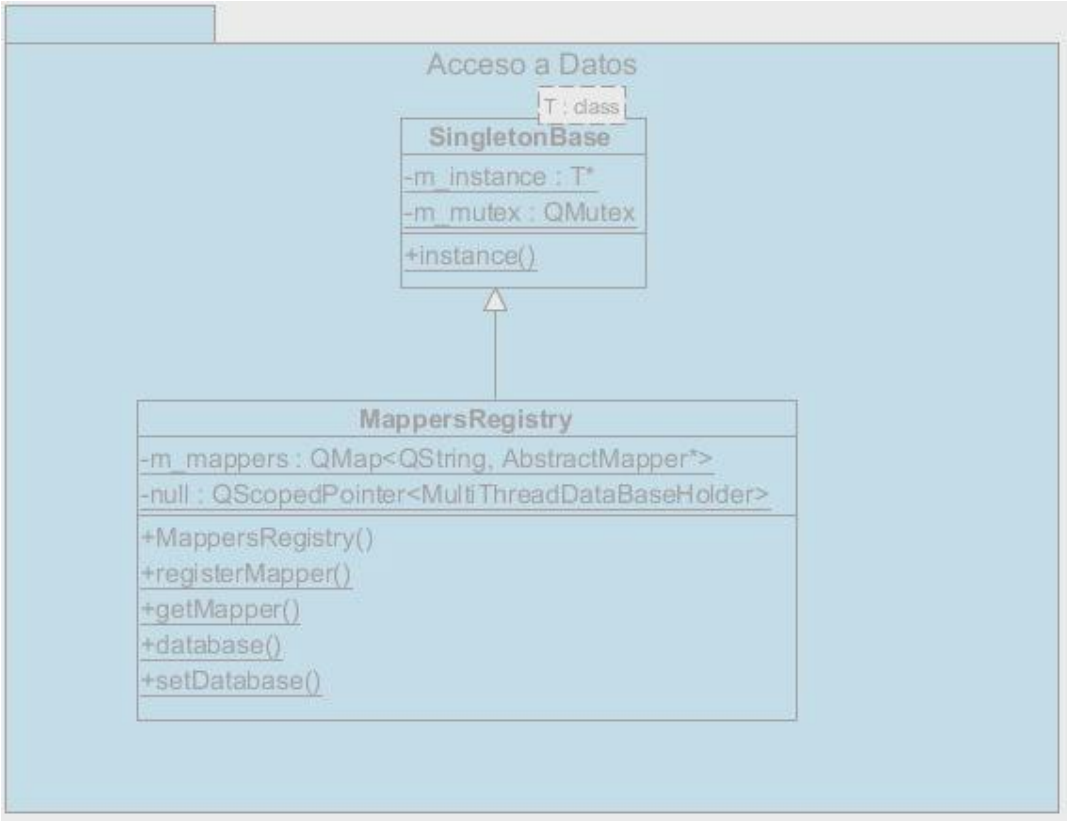


Figura 10: DCD CU Materializar Datos.

El caso de uso Materializar Datos utiliza la clase Mappers_Registry la cual es la encargada de mapear la base de datos y almacenar la información en la memoria de la PC Cliente para su posterior uso, evitando la necesidad de establecer tantas peticiones a la base de datos. Su constructor MappersRegistry() será el encargado de realizar esta tarea. La clase SingletonBase, de la cual hereda MappersRegistry, se encarga de implementar el patrón de diseño que restringe los objetos que se crean pertenecientes a una clase, el mismo garantiza que una clase solamente tenga una instancia y proporciona un punto de acceso global a ella.

3.4.2 Diagrama de clases del diseño para el caso de uso Múltiples conexiones

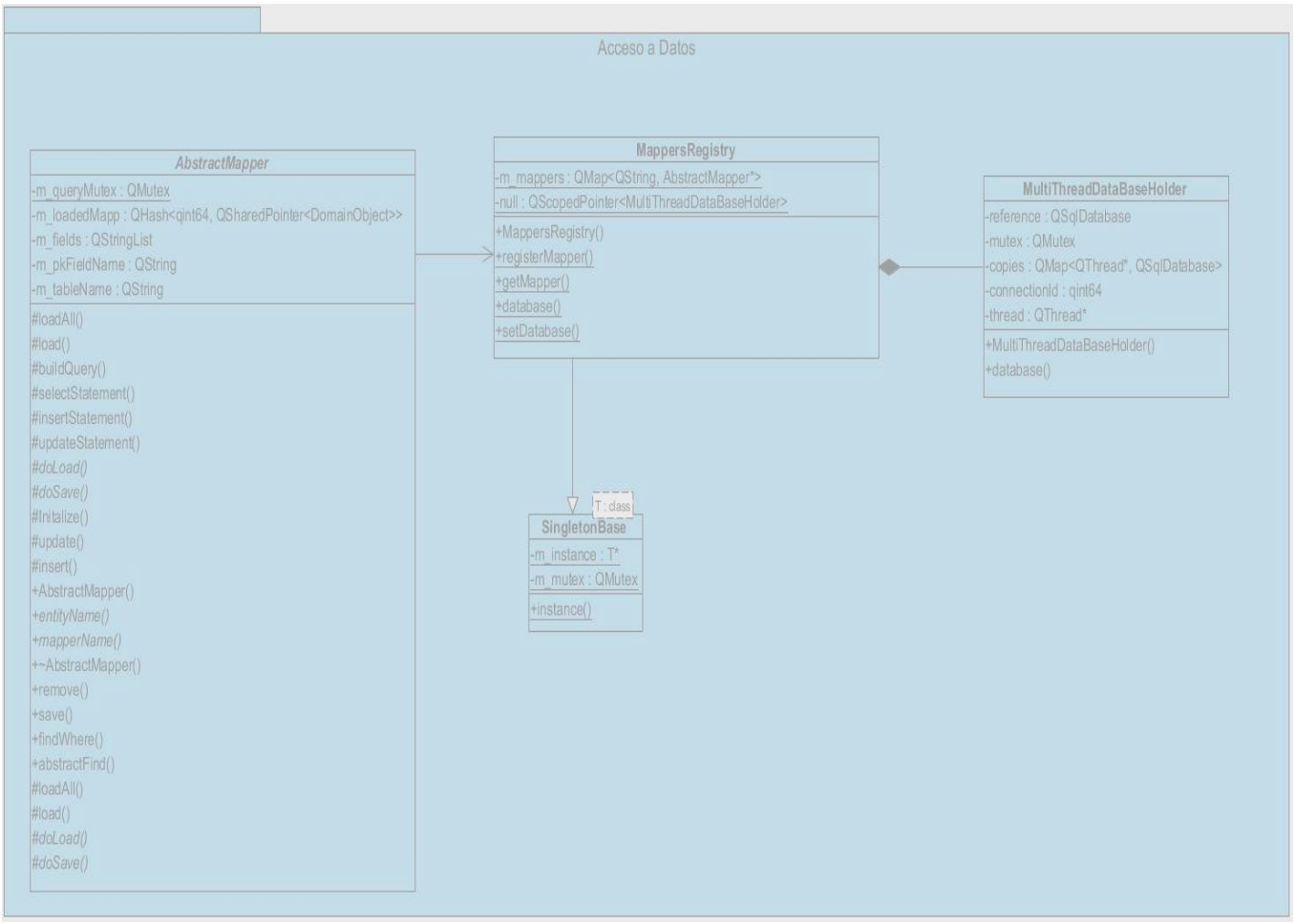


Figura 11: DCD CU Múltiples conexiones.

El caso de uso Múltiples conexiones describe el flujo de realizar varias conexiones a la base de datos desde diferentes hilos de proceso. Se inicializa cuando la aplicación externa realiza una conexión paralela a la base de datos, el sistema a través del método Initalize() de la clase AbstractMapper, accede a la funcionalidad database() de la clase MappersRegistry, esta clase contiene un objeto de la clase MultithreadDataBaseHolder a través del cual accede al método database() de dicha clase que es el responsable de crear una nueva conexión a la base de datos en caso de ser necesario.

3.4.3 Diagrama de clases del diseño para el caso de uso Gestionar Información

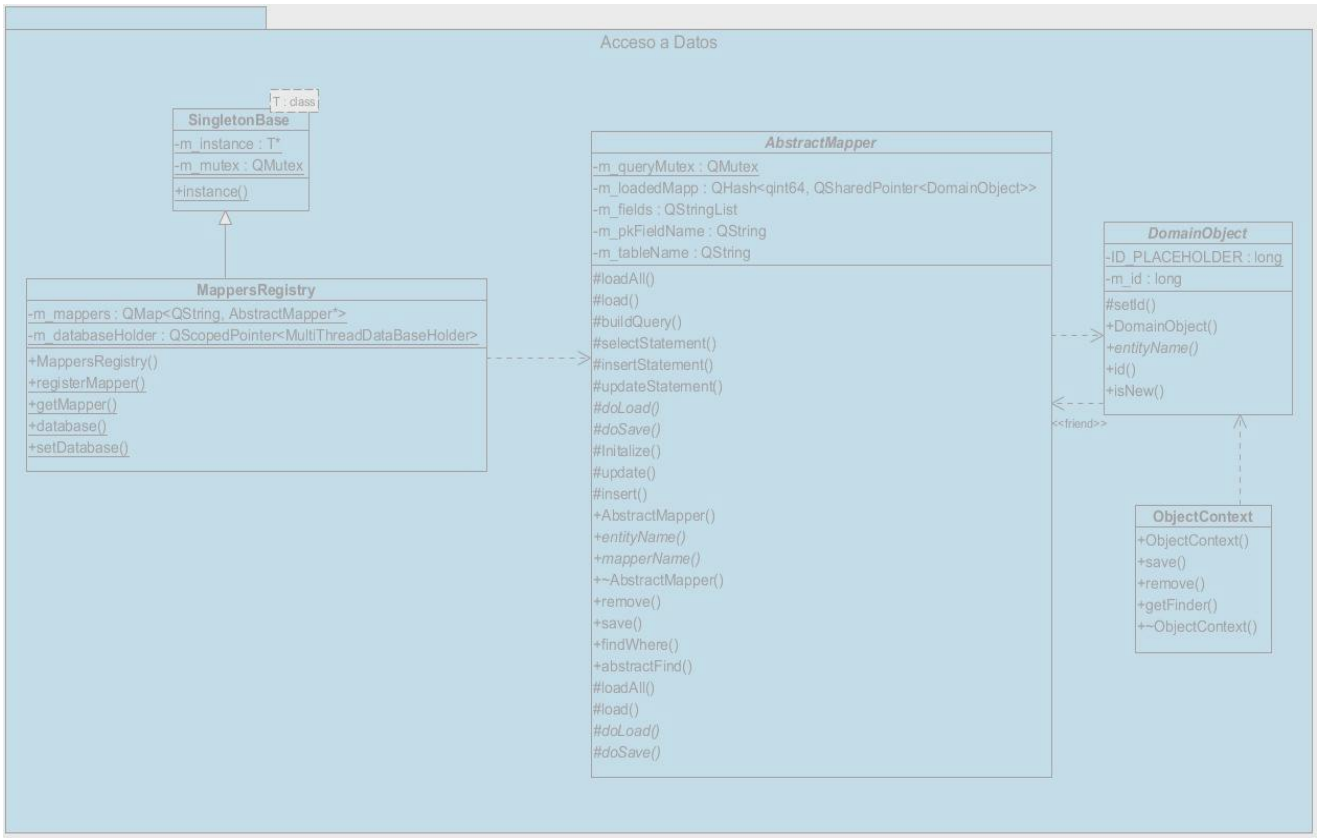


Figura 12: DCD CU Gestionar Información

El caso de uso Gestionar Información describe los procesos de insertar, actualizar y eliminar información en la base de datos. Luego de registrada la base de datos con el método registerMapper() de la clase MappersRegistry, en caso de:

1. Insertar información: una vez creado el objeto a insertar, el sistema a través del método save() de la clase AbstractMapper se encargará de insertar el objeto en la base de datos.
2. Actualizar información: el sistema permite obtener el objeto con los métodos antes mencionados, luego de actualizados los atributos, el sistema guarda los cambios con el método save() de la clase Abstractmapper.

3. Eliminar información: utilizando los métodos findWhere() o abstractFind() de la clase AbstractMapper el sistema permite obtener el objeto a eliminar, luego con el método remove() de la clase AbstractMapper el sistema elimina dicho objeto.

3.4.4 Diagrama de clases del diseño para el caso de uso Desmaterializar Datos

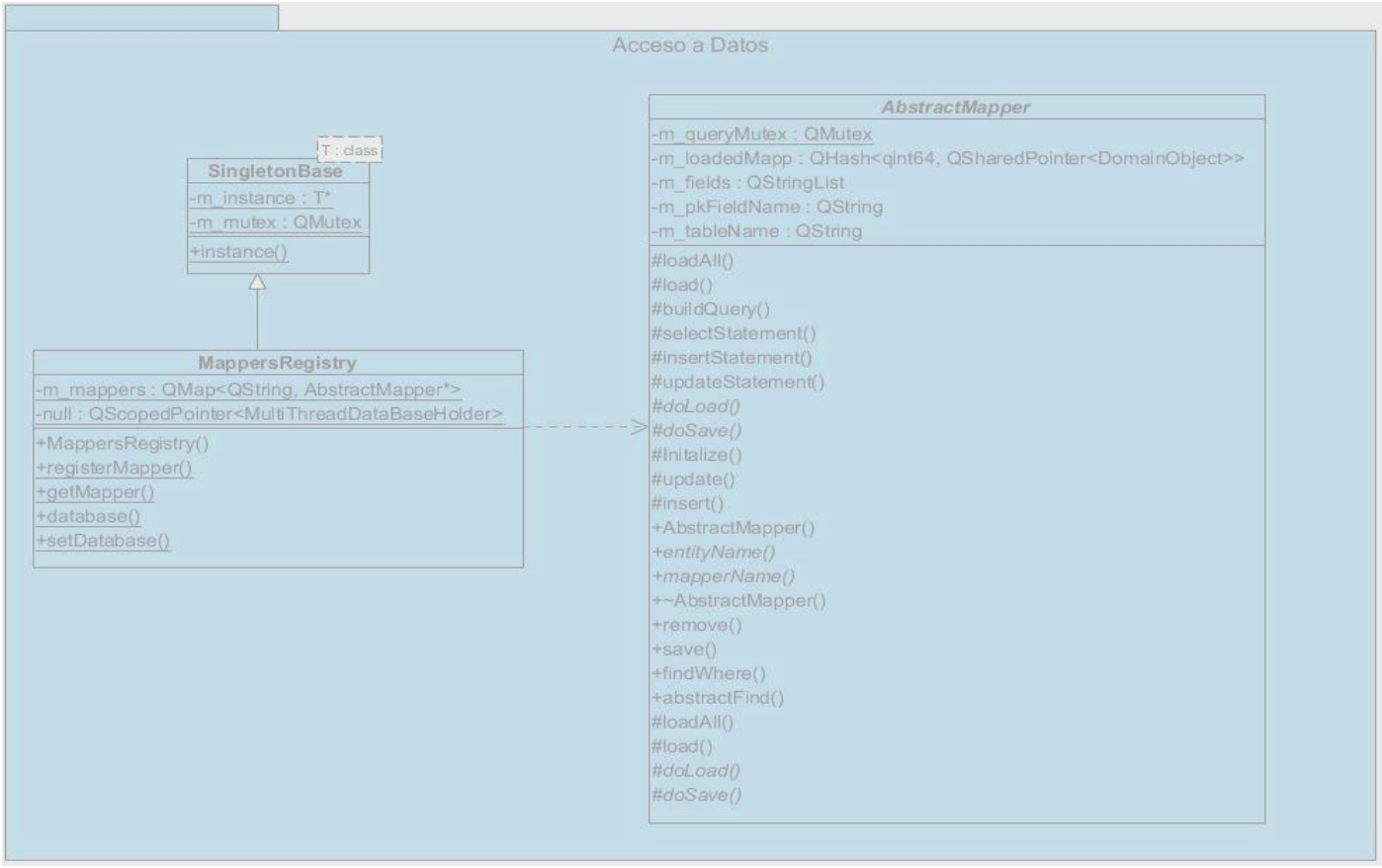


Figura 13: DCD CU Desmaterializar Datos

El caso de uso Desmaterializar Datos muestra el proceso de guardar información en la base de datos desde la programación orientada a objetos. Una vez registrada la base de datos con la clase MappersRegistry. La clase AbstractMapper posee un método doSave() que se encarga de insertar, actualizar o eliminar la información en la base de datos.

3.4.5 Diagrama de clases del diseño para el caso de uso Buscar Objeto

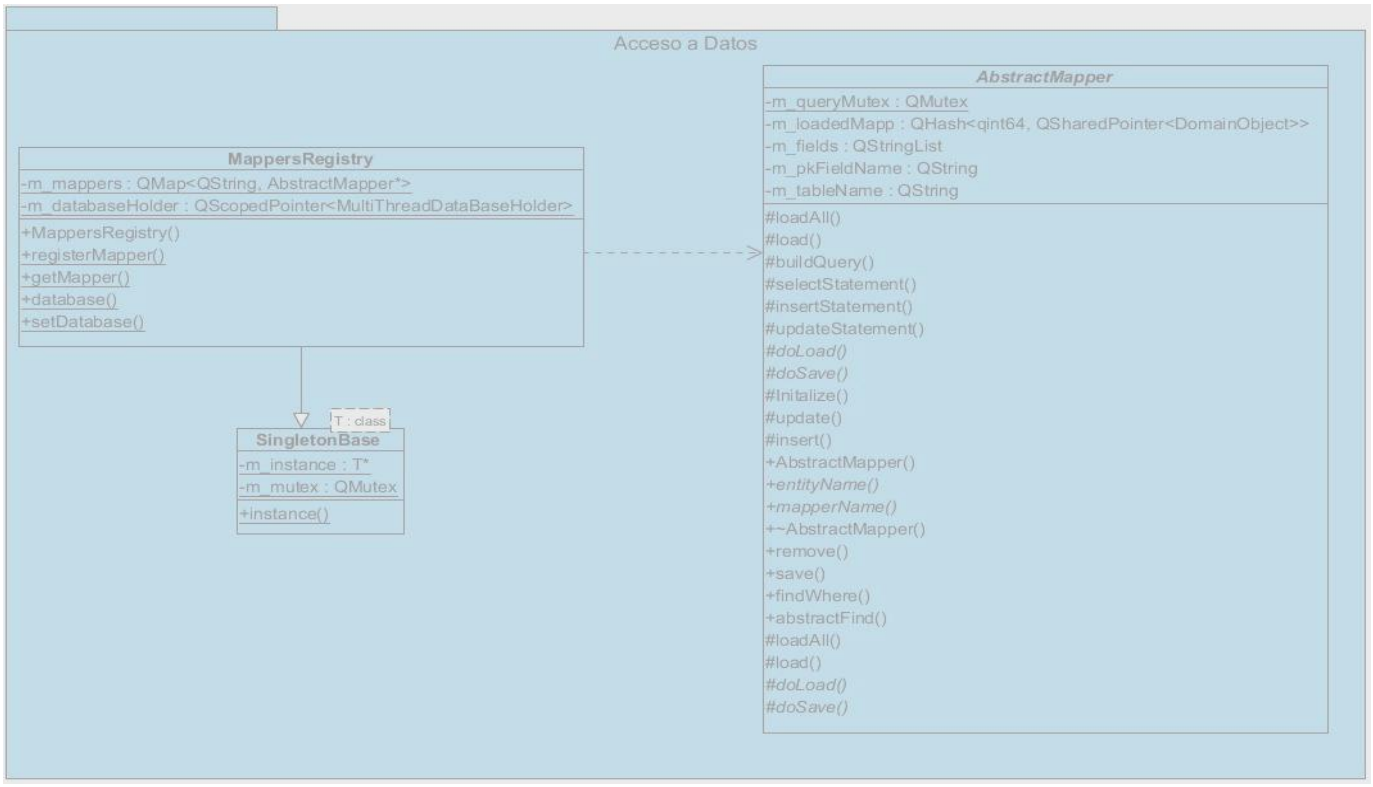


Figura 14: DCD CU Buscar Objeto

El caso de uso Buscar Objeto describe el proceso de respuesta del sistema ante la necesidad de buscar un elemento que se encuentre en la base de datos. Para ello inicialmente se realiza el caso de uso Materializar Datos una vez mapeada la base de datos, mediante la clase AbstractMapper que posee los métodos findById(), findAll(), se realiza la búsqueda en la base de datos.

3.5 Modelo de Despliegue.

Los diagramas de despliegue modelan los aspectos físicos de un sistema, están compuestos por nodos, los mismos encuentran su medio de comunicación en las relaciones que establecen.

Figura 15: Diagrama de Despliegue

3.6 Modelo de Implementación

El modelo de implementación describe la forma en que se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación y en el lenguaje de programación utilizado, y la manera en que dependen los componentes unos de otros. Identifica los componentes físicos de la implementación para que puedan comprenderse y gestionarse mejor**.** (Jacobson, y otros, 2000)

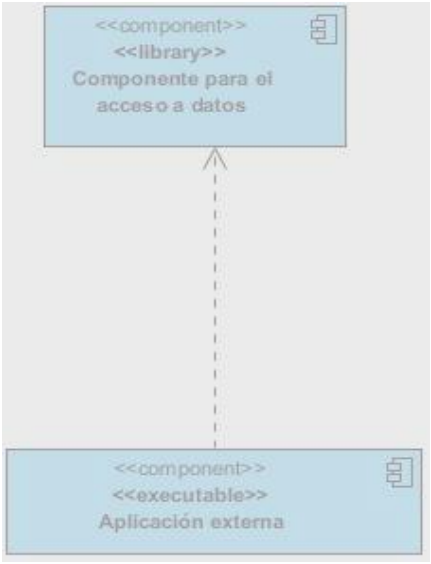


Figura 16: Diagrama de Implementación

3.6.1 Diagrama de Componente

Un componente es una parte física y reemplazable de un sistema, conforma un conjunto de interfaces y las realiza. Los diagramas de componentes modelan los aspectos físicos de un sistema, la vista de

implementación estática de un sistema y los elementos físicos que residen en un nodo, tales como ejecutables, tablas, bibliotecas, archivos y documentos. (Daniele, 2007)

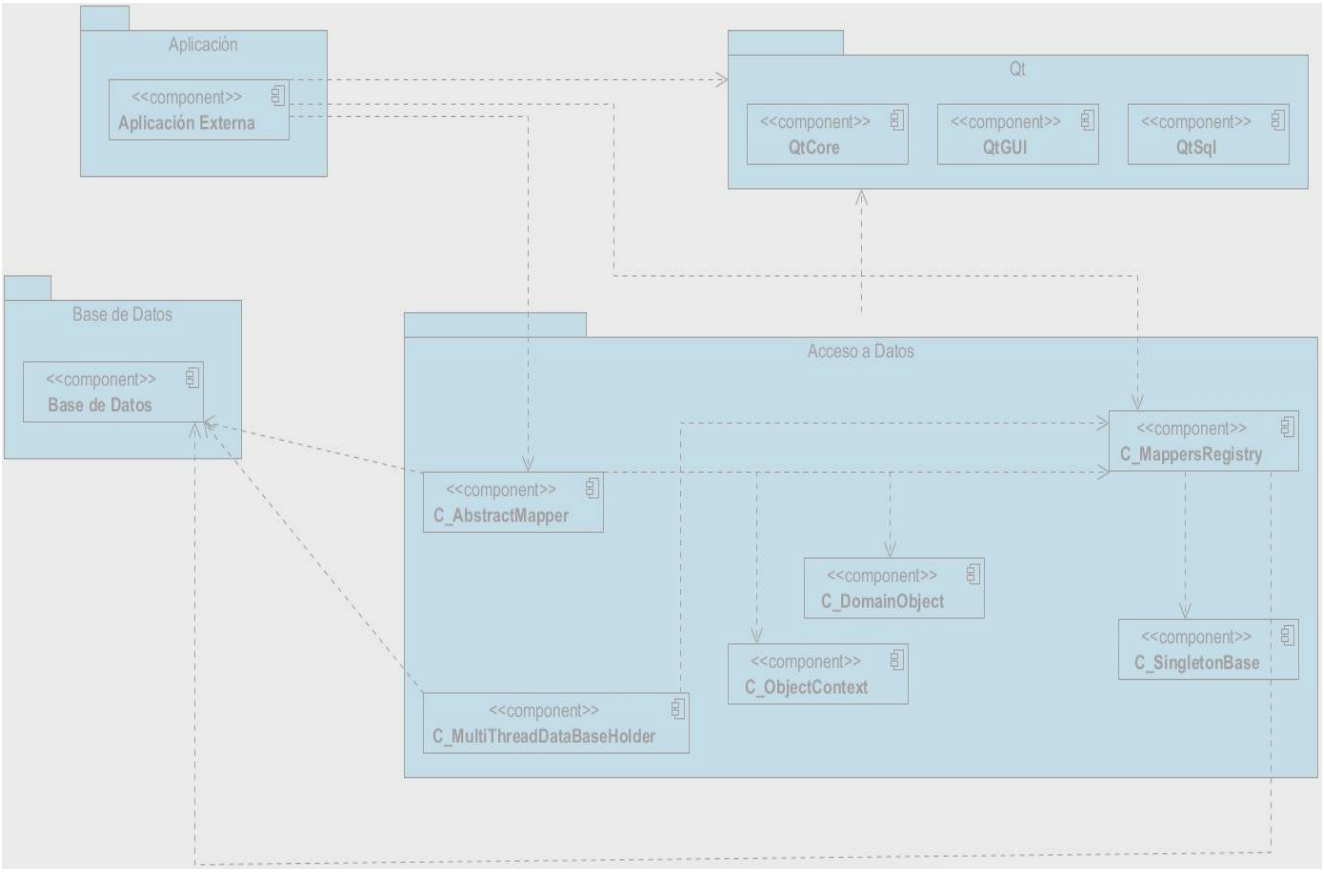


Figura 17: Diagrama de Componente

3.7 Resultados Obtenidos

Con la implementación del componente para el acceso a datos se logra agilizar y facilitar el uso y trabajo con las consultas SQL, logrando que el implementador acceda a la información contenida en la base de datos de una forma sencilla y rápida. Permite que una vez registrada la base de datos al inicio del trabajo, el ORM se encargue de abrir y cerrar estas conexiones tantas veces como sea necesario siéndole invisible al programador esta acción. A continuación se muestran ejemplos de las consultas que se realizan a la base de datos actualmente y en correspondencia las que se realizan con el componente para acceso a datos con soporte multihilo:

Buscar todos los elementos de una tabla:

Desde el *framework* Qt:

```
QSqlQuery *query=new QSqlQuery();
query->exec("SELECT * FROM camera")
```

Desde el componente:

```
QList<Camera*> cam = objMapper->findAll();
```

Buscar un elemento de una tabla dado un Id:

Desde el *framework* Qt:

```
QSqlQuery *query=new QSqlQuery();
query->prepare("SELECT * FROM camera WHERE id_camera=(:id)");
query->bindValue(":id", Id_camera);
query->exec();
```

Desde el componente:

```
Camera *camera = objMapper->findById(id_camera);
```

Para insertar un elemento en la base de datos:

Desde el *framework* Qt:

```
QSqlQuery *query=new QSqlQuery();
query->prepare("INSERT INTO camera(nombre, descripcion)"
"VALUES (:nombre, :descripcion)");
  query->bindValue(0,nombre);
query->bindValue(1,descripcion);
query->exec()
```

Desde el componente:

```
Camera *nueva = new Camera();
```

```
nueva->setNombre("Nombre");
nueva->setDescription("Descripcion");
objMapper->save(nueva);
```

Para actualizar un elemento en la base de datos:

Desde el *framework* Qt:

```
QSqlQuery *query=new QSqlQuery;
query->prepare("UPDATE camera SET nombre=(:nombre) descripcion=(:descripcion)
WHERE id_camera=(:id)");
query->bindValue(":nombre", Axis);
query->bindValue(":descripcion", camara Ip);
query->exec()
```

Desde el componente:

```
Camera *camera = objMapper->findById(id_camera);
camera->setName("Axis");
camera->setDescription("Camara Ip");
objMapper->save(camera);
```

Para eliminar un elemento en la base de datos:

Desde el *framework* Qt:

```
QSqlQuery *query=new QSqlQuery;
query->prepare("DELETE FROM camera WHERE id_camera=(:id)");
query->bindValue(":id", id_camera);
query->exec()
```

Desde el componente:

```
Camera *camera = objMapper->findById(id_camera);
objMapper->remove(camera);
```

La siguiente figura muestra una representación de los ficheros que forman parte del componente para el acceso a datos y que se encargan de implementar las funciones definidas en los requisitos funcionales:

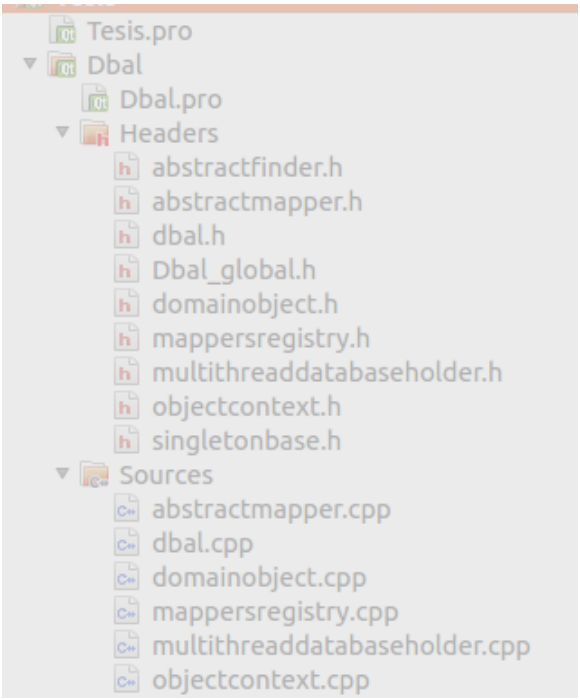


Figura 18: Ficheros que conforman el componente para acceso a datos

3.8 Validación del Sistema

Las pruebas son una actividad en la que un sistema o componente, son ejecutados siguiendo requisitos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema. En la práctica, lo que se prueba puede venir dado por un requisito o colección de requisitos funcionales del sistema cuya implementación justifica una prueba que es posible realizar y que no es demasiado cara de cumplir. Los siguientes son casos de prueba comunes: (Jacobson, y otros, 2000)

- Prueba de caja negra:

Un caso de prueba que especifica como probar un caso de uso o un escenario específico de un caso de uso. Un caso de prueba de este tipo incluye la verificación del resultado de la interacción entre el actor y el sistema, que se satisfacen las precondiciones y poscondiciones especificadas por el caso de uso y que sigue la secuencia de acciones específicas por el caso de uso. Un caso de prueba basado en un caso de

uso especifica típicamente una prueba del sistema como caja negra, es decir, una prueba del comportamiento observable externamente del sistema.
Estas pruebas permiten encontrar:

- Funciones que estén incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.

➢ Prueba de caja blanca:

Un caso de prueba de este tipo puede incluir la verificación de la interacción entre los componentes que implementan un caso de uso. Los casos de prueba basados en la realización de casos de uso, típicamente especifican una prueba del sistema como caja blanca, es decir, una prueba de la interacción interna entre los componentes del sistema.

3.9 Diseño de caso de prueba

Para el caso específico del componente para acceso a datos con soporte multihilo se escogió el método de Prueba de caja negra, debido a que este método permite probar el funcionamiento de los requisitos funcionales y detectar errores de funcionamiento que no fueron detectados durante la etapa de implementación. Las pruebas de caja negra se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se atiene a su especificación. Por ello se denominan pruebas funcionales, y el probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro.

Para la realización de estas pruebas se creó una aplicación externa que es la que toma el rol de actor de sistema, ya que el componente en si no cuenta con una interfaz, la aplicación auxiliar se encarga de probar las funcionalidades que realiza el sistema.

En la fase de prueba se realizaron dos iteraciones, en la primera iteración se encontraron dos no conformidades en el sistema. El caso de uso Gestionar Información no estaba siendo cumplido satisfactoriamente específicamente los requisitos Insertar información y Actualizar información. Por lo que se realizó una segunda iteración de prueba para verificar que se habían eliminado las no conformidades

en este caso de uso, además de verificar el funcionamiento para comprobar que no aparecieran otros problemas en el funcionamiento del sistema. Esta segunda fase concluyó satisfactoriamente ya que no se encontró ningún problema en el componente para acceso a datos con soporte multihilo.

Se muestra una gráfica que representa el comportamiento de los Casos de Uso en sus dos iteraciones del proceso de pruebas:

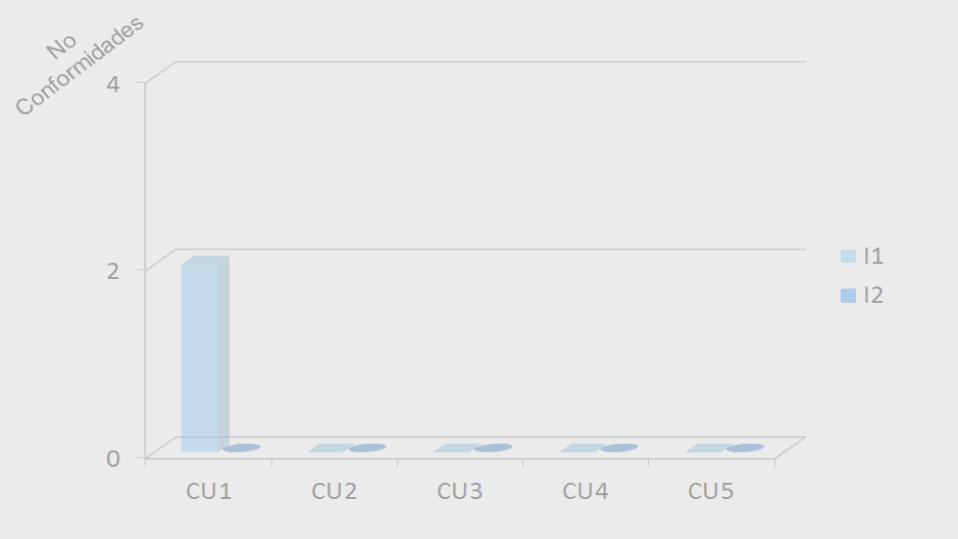


Figura 19: Gráfica que muestra el comportamiento de los Casos de Uso.

A continuación se presentan los casos de pruebas (CP) para los requisitos funcionales de la segunda iteración:

Tabla #1 CP: Gestionar Información

Escenario	**Nombre de sesión**	**Acción Realizada**	**Reacción del sistema**	**Resultado**
Gestionar Información	Insertar Información	La aplicación inserta la información deseada.	El sistema añade la información especificada en las tuplas que forman parte de la base de datos relacional.	satisfactorio

	Actualizar Información	El actor actualiza la información deseada.	El sistema actualiza la información especificada en las tuplas que forman parte de la base de datos relacional	satisfactorio
	Eliminar Información	El actor elimina la información deseada.	El sistema elimina la información especificada en las tuplas que forman parte de la base de datos relacional.	satisfactorio

Tabla #2 CP: Buscar Objeto

Escenario	**Nombre de sesión**	**Acción Realizada**	**Reacción del sistema**	**Resultado**
Buscar Objeto	Buscar un objeto dado un criterio.	El actor escribe los parámetros de la búsqueda.	El sistema de devuelve los datos buscados	satisfactorio
	Buscar todos los objetos	El actor invoca el método para buscar todos los objetos.	El sistema de devuelve los datos buscados	

Para los casos de uso Materializar datos y Desmaterializar datos se prueban con el funcionamiento de los dos casos de uso anteriores, ya que sin ellos no sería posible la comunicación entre la base de datos y la aplicación.

3.10 Conclusiones

En el capítulo se generaron los artefactos necesarios para el análisis y diseño del sistema los mismos son un plano que constituye la guía para la implementación. Como conclusiones parciales se pueden arribar a las siguientes:

- El diagrama de componentes realizado describe la estructura física del sistema mediante las relaciones entre sus componentes. De esta manera se obtiene una vista de implementación estática del sistema.
- La utilización de los patrones instancia única, mapa identidad, registro y *data mapper* permiten flexibilizar la implementación del sistema en función de los objetivos planteados.
- La aplicación de pruebas de caja negra al sistema permitió comprobar su correcto funcionamiento, certificando que cumple los requisitos funcionales y no funcionales que fueron definidos previamente. Se realizaron cinco casos de prueba, asociando uno a cada caso de uso del sistema.

Conclusiones

Como resultado de la investigación se obtuvo el componente para el acceso a datos con soporte multihilo para el departamento Señales Digitales. El sistema desarrollado permite realizar consultas a la base de datos sin necesidad de utilizar las tradicionales consultas SQL. Una vez concluida la investigación se concluye que:

- El estudio de los conceptos asociados al problema planteado posibilitó el establecimiento de las bases teóricas para la investigación realizada. A partir del análisis de las soluciones existentes, surgió como propuesta de solución la confección del componente para el acceso a datos con soporte multihilo.

- La elección de la arquitectura basada en componentes ayudó a obtener un producto final de mayor calidad, teniendo en cuenta que al ser dividido en componentes cada uno podrá ser construido y mejorado de forma individual. De esta manera se simplifica el mantenimiento del sistema y se logra una mayor reutilización del software.

- La estrategia de prueba definida incluye la realización de cinco casos de prueba de caja negra correspondientes a cada uno de los casos de uso del sistema. Los resultados obtenidos mediante su aplicación al componente permitieron validar la completitud de respuesta a los requerimientos especificados durante la fase inicial del proceso de desarrollo.

- Con la confección del componente para acceso a datos se permite el mapeo de los objetos desde la programación orientada a objetos a las bases de datos relacionales, mejorando así la capa de acceso a datos del departamento Señales Digitales.

Al concluir el presente trabajo se recomienda:

- Establecer una estrategia para liberar espacio en la memoria RAM cuando se cargan los datos de la base de datos ya que el rellenado de la misma puede ralentizar el trabajo.
- Crear componentes para buscar dado determinados atributos de los objetos.

Bibliografía Referenciada

Alberca, Alejandro Manzaneque y Galvez, Jesús Díaz-Tendero. 2011. *Bases de datos Orientadas a Objetos y Bases de Datos Objeto-Relacionales.* 2011.

Alonso, Evelyn Menéndez. 2009. *Herramientas CASE para el proceso de desarrollo de Software.* 2009.

Aros, Celio Gil. 2008. *RUP: Metodología en los sistemas y aplicaciones basadas en la web.* 2008.

Bases de Datos Relacional. **Caviedes, Diego Fernando Ortega y Vives Santodomingo, Pedro Luis. 2011.** 2011.

Computación, Departamento de Sistemas Informáticos y. 2012. Departamento de Sistemas Informáticos y Computación. *Departamento de Sistemas Informáticos y Computación.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://users.dsic.upv.es/~jorallo/docent/BDA/castella/tema2d.pdf. ISSN.

Cursodejava. 2012. Cursodejava. *Cursodejava.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://cursodejava.com.mx/cursodejava102.html. ISSN.

Cursos. 2012. cs.us. *cs.us.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://www.cs.us.es/cursos/bd-2001/temas/sql_I.html. ISSN.

desarrolladoresweb. 2012. desarrolladoresweb. *desarrolladoresweb.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://www.desarrolladoresweb.com/articuos/499.php. ISSN.

Diagramas de Casos de Uso. **Tello, Jesús Cáceres.**

Espinoza, Guillermo. 2012. ProAndes CFT. *ProAndes CFT.* [En línea] 5 de Diciembre de 2012. [Citado el: 5 de Diciembre de 2012.] www.soeduc.cl/apuntes/basededatos.doc. ISSN.

Gutiérrez, Javier J. 2012. Departamento de Lenguajes y Sistemas Informáticos. *Departamento de Lenguajes y Sistemas Informáticos.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf. ISSN.

Isaías Carrillo Pérez y Pérez González, Rodrigo. 2008. *Las metodologías orientadas al control de los procesos, estableciendo .* 2008.

Jacobson, Ivar y Rumbaugh, James. 2000. *El Proceso de Desarrollo de Software.* 2000.

Laguna, Miguel A. 2011. *Ingeniería del Software I.* 2011.

Larman, Craig y Hall, Prentice. 2003. *UML y Patrones. 2ª Edición.* 2003.

León, Welicki. 2005. msdn. [En línea] microsoft, 2005. http://msdn.microsoft.com/es-es/library/bb972272.aspx.

Mapeo del modelo de objetos al modelo relacional . **Pizarro, Pablo. 2005.** Mendoza : s.n., 2005.

Marcos, Esperanza. 2005. *Diseño de bases de datos objeto-relacionales con UML.* España : s.n., 2005.

Màrius, Armenteras Comellas. 2007. 2007.

Martin Fowler . 2001. Martin Fowler . [En línea] 2001. http://martinfowler.com/eaaCatalog/index.html.

Mastermagazine. 2012. Mastermagazine. *Mastermagazine.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://www.mastermagazine.info/termino/6771.php. ISSN.

Modelado Básico con Casos de Uso. **Universidad Carlos III de Madrid.** Madrid : s.n.

Object Relational Mapping. **Escuela de Ingeniería de Sistemas y Computación. 2008.** 2008.

Orallo, Enrique Hernández. 2002ojo. *El Lenguaje Unificado de Modelado (UML).* 2002ojo.

Paredes, Adrián. 2008. *El ciclo de vida de un Mapa de Identidad es una transacción de negocio.* 2008.

Pérez, Laura Grandes y Vaquero, Leonorl Borrego. 2012. wikispace.com. *wikispace.com.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://topicosjvcarlos.wikispaces.com/file/view/bdoo.pdf. ISSN.

Pimentel, Ernesto. 2007. *Patrones de diseño.* 2007.

QDjango. 2012. QDjango. *QDjango.* [En línea] 4 de diciembre de 2012. [Citado el: 2012 de diciembre de 2012.] http://code.google.com/p/qdjango/.

QxORM. 2012. QXORM. *QXORM.* [En línea] 5 de diciembre de 2012. [Citado el: 5 de diciembre de 2012.] http://www.qxorm.com/qxorm_en/home.html. ISBN.

Reporte de instalación de apache. **Can, Carolina Novelo. 2010.** 2010.

Romero, Lisandra Díaz y Sardiñas, Yosbel Marín. 2010. biblioteca.uci.cu. *biblioteca.uci.cu.* [En línea] 4 de diciembre de 2010. [Citado el: 4 de diciembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_03573_10. ISBN.

Ruiz, Isnardo Reducindo. 2009. *Programación Multihilos.* 2009.

Sánchez, Jorge. 2004. *Principios sobre bases de datos relacionales.* 2004.

Synthesis, Code. 2012. Code Synthesis. *Code Synthesis.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de 2012 de 2012.] http://www.codesynthesis.com/products/odb/. ISSN.

techerald.com. 2010. Techerald. *Techerald.* [En línea] 2010. [Citado el: 1 de noviembre de 2012.] http://techerald.com/page/que-es-qt-ejemplo-clasico-hola-mundo--230320090143.html..

Terreros, Julio Casal. 2013. msdn. [En línea] microsoft, 2013. [Citado el: 10 de 04 de 2013.] http://msdn.microsoft.com/es-es/library/bb972268.aspx.

Tore, Horacio Aldo. 2010. *Patrones de diseño de software.* 2010.

Torossi, Gustavo. 2008. *El Proceso Unificado de Desarrollo de Software.* 2008.

Zaninotto, Francois y Potencier, Fabien. 2008. *Symfony la guía definitiva.* 2008.

Zator Systems. 2012. Zator Systems: Tecnología de la información para el conocimiento. *Zator Systems: Tecnología de la información para el conocimiento.* [En línea] 2012. [Citado el: 5 de Noviembre de 2012.] http://www.zator.com/Cpp/E1_2.htm.

Lima Mena, Anier y Enamorado Zamora, Yanara. 2010. *Perfeccionamiento de la capa de acceso a datos del HelpDesk.* 2010.

Alberca, Alejandro Manzaneque y Galvez, Jesús Díaz-Tendero. 2011. *Bases de datos Orientadas a Objetos y Bases de Datos Objeto-Relacionales.* 2011.

Alonso, Evelyn Menéndez. 2009. *Herramientas CASE para el proceso de desarrollo de Software.* 2009.

Aros, Celio Gil. 2008. *RUP: Metodología en los sistemas y aplicaciones basadas en la web.* 2008.

Bases de Datos Relacional. **Caviedes, Diego Fernando Ortega y Vives Santodomingo, Pedro Luis. 2011.** 2011.

Computación, Departamento de Sistemas Informáticos y. 2012. Departamento de Sistemas Informáticos y Computación. *Departamento de Sistemas Informáticos y Computación.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://users.dsic.upv.es/~jorallo/docent/BDA/castella/tema2d.pdf. ISSN.

Cursodejava. 2012. Cursodejava. *Cursodejava.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://cursodejava.com.mx/cursodejava102.html. ISSN.

Cursos. 2012. cs.us. *cs.us.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://www.cs.us.es/cursos/bd-2001/temas/sql_I.html. ISSN.

Daniele, Marcela. 2007. *EL ARTE DE MODELAR.* 2007.

desarrolladoresweb. 2012. desarrolladoresweb. *desarrolladoresweb.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://www.desarrolladoresweb.com/articuos/499.php. ISSN.

Diagramas de Casos de Uso. **Tello, Jesús Cáceres.**

Díaz Romero, Lisandra y Marín Sardiñas, Yosbel. 2010. *Framework de Mapeo Objeto- Relacional para PHP.* 2010.

Díaz, Milenis Fernández. 2011. *Componente para la abstracción de la capa de acceso a datos del sistema GeolMin.* 2011.

Dirección General del Gobierno Digital. 2006. *Estándares de codificación de sistemas.* 2006.

Espinoza, Guillermo. 2012. ProAndes CFT. *ProAndes CFT.* [En línea] 5 de Diciembre de 2012. [Citado el: 5 de Diciembre de 2012.] www.soeduc.cl/apuntes/basededatos.doc. ISSN.

Flower, Martin, y otros. 2002. *Patterns of Enterprise Application Architecture.* 2002.

Gutiérrez, Javier J. 2012. Departamento de Lenguajes y Sistemas Informáticos. *Departamento de Lenguajes y Sistemas Informáticos.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf. ISSN.

Isaías Carrillo Pérez y Pérez González, Rodrigo. 2008. *Las metodologías orientadas al control de los procesos, estableciendo .* 2008.

Jacobson, Ivar y Rumbaugh, James. 2000. *El Proceso de Desarrollo de Software.* 2000.

Laguna, Miguel A. 2011. *Ingeniería del Software I.* 2011.

Larman, Craig y Hall, Prentice. 2003. *UML y Patrones. 2ª Edición.* 2003.

León, Welicki. 2005. msdn. [En línea] microsoft, 2005. http://msdn.microsoft.com/es-es/library/bb972272.aspx.

López Miláan, Alberto y Chung La Rosa, Edgardo Juan. 2009. *Implementación de la capa de persistencia de datos de los módulos de Presentación y Contratación del proyecto Convenio Integral de Cooperación Cuba Venezuela.* 2009.

Luis Fernández, Elian y López Naranjo, Danis. 2007. *Desarrollo de la capa de acceso a datos para los modulos de administración y nomencladoresdel Sistema de Gestión de Inventario de Almacenes.* 2007.

Mapeo del modelo de objetos al modelo relacional . **Pizarro, Pablo. 2005.** Mendoza : s.n., 2005.

Marcos, Esperanza. 2005. *Diseño de bases de datos objeto-relacionales con UML.* España : s.n., 2005.

Màrius, Armenteras Comellas. 2007. 2007.

Martin Fowler . 2001. Martin Fowler . [En línea] 2001. http://martinfowler.com/eaaCatalog/index.html.

Mastermagazine. 2012. Mastermagazine. *Mastermagazine.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://www.mastermagazine.info/termino/6771.php. ISSN.

Modelado Básico con Casos de Uso. **Universidad Carlos III de Madrid.** Madrid : s.n.

Object Relational Mapping. **Escuela de Ingeniería de Sistemas y Computación. 2008.** 2008.

Orallo, Enrique Hernández. 2002ojo. *El Lenguaje Unificado de Modelado (UML).* 2002ojo.

Paredes, Adrián. 2008. *El ciclo de vida de un Mapa de Identidad es una transacción de negocio.* 2008.

Pérez, Laura Grandes y Vaquero, Leonorl Borrego. 2012. wikispace.com. *wikispace.com.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de diciembre de 2012.] http://topicosjvcarlos.wikispaces.com/file/view/bdoo.pdf. ISSN.

Pimentel, Ernesto. 2007. *Patrones de diseño.* 2007.

QDjango. 2012. QDjango. *QDjango.* [En línea] 4 de diciembre de 2012. [Citado el: 2012 de diciembre de 2012.] http://code.google.com/p/qdjango/.

QxORM. 2012. QXORM. *QXORM.* [En línea] 5 de diciembre de 2012. [Citado el: 5 de diciembre de 2012.] http://www.qxorm.com/qxorm_en/home.html. ISBN.

Reporte de instalación de apache. **Can, Carolina Novelo. 2010.** 2010.

Romero, Lisandra Díaz y Sardiñas, Yosbel Marín. 2010. biblioteca.uci.cu. *biblioteca.uci.cu.* [En línea] 4 de diciembre de 2010. [Citado el: 4 de diciembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_03573_10. ISBN.

Ruiz, Isnardo Reducindo. 2009. *Programación Multihilos.* 2009.

Sánchez, Jorge. 2004. *Principios sobre bases de datos relacionales.* 2004.

Synthesis, Code. 2012. Code Synthesis. *Code Synthesis.* [En línea] 4 de diciembre de 2012. [Citado el: 4 de 2012 de 2012.] http://www.codesynthesis.com/products/odb/. ISSN.

techerald.com. 2010. Techerald. *Techerald.* [En línea] 2010. [Citado el: 1 de noviembre de 2012.] http://techerald.com/page/que-es-qt-ejemplo-clasico-hola-mundo--230320090143.html..

Terreros, Julio Casal. 2013. msdn. [En línea] microsoft, 2013. [Citado el: 10 de 04 de 2013.] http://msdn.microsoft.com/es-es/library/bb972268.aspx.

Tore, Horacio Aldo. 2010. *Patrones de diseño de software.* 2010.

Torossi, Gustavo. 2008. *El Proceso Unificado de Desarrollo de Software.* 2008.

Zaninotto, Francois y Potencier, Fabien. 2008. *Symfony la guía definitiva.* 2008.

Zator Systems. 2012. Zator Systems: Tecnología de la información para el conocimiento. *Zator Systems: Tecnología de la información para el conocimiento.* [En línea] 2012. [Citado el: 5 de Noviembre de 2012.] http://www.zator.com/Cpp/E1_2.htm.

Componente para el acceso a datos: Conjunto de clases implementadas para obtener la información desde el modelo relacional y traducirla al modelo orientado a objetos y de forma inversa. Capa intermedia entre la programación y la base de datos.

Materializar: Obtener los datos desde el modelo relacional y traducirlos a datos en el modelo orientado a objetos para trabajar con ellos desde la aplicación externa.

Desmaterializar: Obtener los datos desde el modelo orientado a objetos y traducirlos a datos en el modelo relacional para insertarlo en la base de datos.

Multiplataforma: Capacidad de una aplicación o herramienta de trabajar sin importar el sistema operativo (Linux o Windows).

Multihilo: Capacidad del componente para el acceso a datos de realizar varias consultas a la base de datos de forma simultánea.

Objetos clases persistentes: Objetos que van a persistir en las tuplas de las tablas de la base de datos relacional.

Estándar de facto: Es aquel patrón o norma que se caracteriza por no haber sido consensuada ni legitimada por un organismo de estandarización al efecto.