Universidad de las Ciencias Informáticas

Facultad 5

Centro de Informática Industrial.

Manejador para la comunicación con sistemas de bases de datos.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Autor

Eduardo Rafael Hung Gutiérrez.

Tutor

Ing. Yunier Velázquez Batista.

“Ciudad de la Habana, Julio 2010“

“Año 52 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaro ser autor de este trabajo de Diploma y reconocemos a la Universidad de las Ciencias Informáticas los derechos primordiales del mismo con carácter exclusivo.

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

Eduardo R. Hung Gutiérrez

_____

FIRMA DEL AUTOR

Ing. Yunier Velázquez Batista

_____

FIRMA DEL TUTOR

DATOS DE CONTACTO

Ing. Yunier Velázquez Batista.

Graduado de Ingeniero en Informática en el 2006, profesor de la Universidad de las Ciencias Informáticas con categoría docente de Instructor, con tres años de experiencia vinculado a la docencia y a la producción de software en el Centro de Informática Industrial.
Universidad de las Ciencias Informáticas, Habana, Cuba.

Email: yunierve@uci.cu

AGRADECIMIENTOS

Agradecer primeramente a mi mamá por ser la persona que se ha sacrificado por mí en los momentos claves de mi vida para que este momento llegara. Por haberme inculcado la necesidad de estudiar una carrera universitaria y por supuesto enseñarme a dar lo mejor de mí. De manera especial a mi abuelita Hilda que ha sido para mí mas que mi madre. A ella por haber confiado en mí, cuando muchas personas dudaban de mi capacidad, por aconsejarme y darme fuerzas antes de iniciar una prueba. Todo lo que en mi persona esta se lo debo a ella, por lograr instruirme durante mi educación gran parte de las cosas que hoy me rigen. A mi abuelo Agustín por ser en gran parte ejemplo para mí, a ese que me enseño también muchas cosas buenas durante mi educación. A mi tía Isaira que aunque este en Venezuela, siempre la llevaré en mi mente, porque ha sido una persona a la cual he querido durante muchos años y aun la sigo queriendo. A ella le debo también gran parte de mí, sin ella no estuviese aquí, en esta Universidad y a ella le debo una parte de mi educación. A mi hermana Lien, la cual aún siendo un poco más joven que yo, ha sido mi ídolo. Ella me ha enseñado que si se puede estudiar y sacar buenas notas a pesar de muchas cosas y espero que ella siga mis pasos. A mis primos-hermanos Reynaldo y Reinier por ser parte de mí también, a ellos los quiero y al igual que mi hermana espero que sigan mis pasos.

A mi novia Yinet Garbey que más que mi novia, ha sido mi amiga, mi hermana, mi compañera. A ella que ha sabido darme todo su amor, su cariño, su comprensión, y su apoyo durante todos estos 5 años de carrera universitaria y por supuesto por soportarme de la mejor manera. Sin ti creo que no hubiese podido lograr todo lo que he hecho hasta hoy, y decir que has logrado instruirme muchos valores que para mí no eran visibles en un principio.

A todos mis hermanos y amigos del llamado "Cartel" a Yoanni, Joel, Hermes, Causse, Geovanis, Santiago, Yelena, Walny, Fernando, que estoy seguro que sin ellos no me hubiese sido un poco más placentera la vida aquí en la UCI. A todos ellos les debo muchas cosas que lograron

enseñarme también, como es el saber apreciar una amistad, una hermandad, un compañerismo. A todos ellos muchas gracias por ser parte de mí y por soportarme al igual que mi novia.

A mi Tutor, el profe Yunier, que desde un principio me apoyó en el desarrollo de este Trabajo de Diploma y me obligó a dar lo mejor de mí y a prepararme mejor cada día más.

A todos mis compañeros de aula con quienes he compartido todos estos años, mi tiempo de estudiante. En especial a Yisel, Adys, Karen, Yamilet por ser buenas compañeras.

A Bárbara, Lisandra, Islema por ser más que compañeras, buenas amigas.

A dos amigos que para mí significan algo: a Dayron que aunque no este con nosotros en la escuela, sigue siendo un amigo más y a Yadira O´Reilly que ha sido para mí una compañera y una buena amiga a pesar de muchas cosas.

A Yalbert y al Riqui que han sido buenos amigos aunque tengamos rivalidad en el Fútbol.

De manera general a todos mis amigos y amigas que de una forma u otra han formado parte de mí.

A todos los profesores que contribuyeron en mi formación desde mi niñez, a todos llegue mi más sincero agradecimiento.

DEDICATORIA

A mi madre Celeste.

A mis abuelos Hilda y Agustín.

A mi tía Isaira.

A mi hermana Lien y a mis primos Reynaldo y Reinier.

Al resto de mi familia y amigos en general.

RESUMEN

El presente trabajo de diploma describe el desarrollo de un manejador para la comunicación con Sistemas Gestores de Bases de Datos. El desarrollo del mismo surge a partir de que el módulo de adquisición de datos del sistema de Supervisión, Control y Adquisición de Datos del Centro de Informática Industrial (CEDIN), perteneciente a la Universidad de las Ciencias Informáticas, no cuenta con un mecanismo de adquisición de datos que permita la comunicación con Sistemas Gestores de Bases de Datos, existiendo únicamente la comunicación con dispositivos de campo a través de protocolos industriales.

Este manejador (en inglés, *driver*) está encapsulado en una biblioteca de carga dinámica que será utilizada por el sistema de Adquisición, Supervisión y Control de Datos; del CEDIN, lo cual le permitirá recolectar datos existentes en bases de datos. Se utiliza como biblioteca de acceso a datos SOCI, la cual le permite acceder a las bases de datos de manera transparente a través de consultas SQL. Esta biblioteca brinda soporte para la comunicación con Gestores de Bases de Datos más utilizados en la actualidad, como son: PostgreSQL, MySQL y Oracle; aunque en el trabajo la solución se orienta a PostgreSQL.

La primera etapa de este trabajo consistió en el estudio de los conceptos fundamentales contenidos dentro de los procesos de adquisición de datos en los sistemas de automatización industrial. Se analizan las características principales y elementos actuales de los mismos, así como los manejadores de dispositivos en particular.

La etapa siguiente se centró en un proceso de selección de tecnologías y herramientas de desarrollo para el análisis, diseño e implementación del manejador. Se seleccionaron los requisitos que forman parte del diseño e implementación de las capas más esenciales que conforman el manejador. Por último, se realizaron las pruebas para la validación de los requisitos.

PALABRAS CLAVES

Sistemas Gestores de Bases de Datos, adquisición de datos, dispositivos de campo, manejadores de dispositivos

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El siglo XXI es conocido por los expertos como “el inicio de la Era de la Información”. Es la etapa del desarrollo de la humanidad en la que la globalización de los conocimientos y el dinamismo constante de las tecnologías son un hecho. Estas tecnologías hoy en día juegan un rol importante dentro del desarrollo de un país. La naciente industria cubana del software cuenta con muchos retos sociales y económicos para acelerar la informatización de la sociedad cubana.

La Universidad de las Ciencias Informáticas (UCI) forma parte de este desarrollo, cuya misión es formar profesionales comprometidos con la Revolución, partiendo de un modelo pedagógico flexible, que vincula dinámicamente el estudio con la producción y la investigación. La misma promueve la investigación de tecnologías novedosas, metodologías de trabajo internacionales acreditadas puestas en práctica y la creación de soluciones integrales en la producción de software, donde el Centro de Informática Industrial (CEDIN) toma participación y protagonismo; siendo una de sus líneas de productos el desarrollo de sistemas SCADA *(Supervisory Control And Data Adquisition).*

Un sistema SCADA es un sistema de adquisición, supervisión y control de datos, que permite la comunicación con dispositivos de campo (controladores autónomos, autómatas programables, etc.). Dentro de las características importantes en estos sistemas está la fácil adaptación a múltiples entornos; así como la posibilidad de integración con otros sistemas, ya sean gerenciales, de gestión u otros.

Los sistemas SCADA deben cumplir con un conjunto de requerimientos generales, y en muchos casos específicos según la finalidad con que son diseñados. La solución SCADA “Guardián del ALBA” desarrollada en el CEDIN posee como principales funciones: el procesamiento y almacenamiento de datos históricos; manejo de alarmas, comandos, eventos así como el diseño y visualización de gráficos, reportes, etc. Para que todo esto sea posible el mismo realiza además la adquisición de datos desde dispositivos de campos a través de los manejadores. Para el intercambio de información con otros sistemas, ya sean de automatización o gestión, se ha hecho viable la utilización de Bases de Datos como fuentes comunes. Actualmente los manejadores desarrollados para el proyecto SCADA “Guardián del ALBA” están orientados a protocolos industriales, quedando de esta forma limitada la adquisición de datos desde otras fuentes de información, como son los Sistemas Gestores de Bases de Datos Relacionales (RDBMS).

Es por ello que se formuló el siguiente **problema científico**: ¿Cómo posibilitar al sistema SCADA “Guardián del ALBA” la adquisición de datos mediante la comunicación con Sistemas Gestores de Base de Datos?

A su vez el **objeto de estudio** se centra en: mecanismos y componentes para la comunicación con Sistemas de Bases de Datos y como **campo de acción**: manejador para la comunicación con Sistemas de Bases de Datos en el proyecto SCADA “Guardián del ALBA”.

El **objetivo general** de este trabajo consiste en: desarrollar un manejador para la comunicación con Sistemas de Bases de Datos, que cumpla con la Interfaz Genérica de los Manejadores del proyecto SCADA “Guardián del ALBA”.

Del objetivo general planteado anteriormente se derivan las siguientes **tareas investigativas**:

- ✓ Estudio y análisis de la bibliografía que hace referencia a los Sistemas Gestores de Bases de Datos.
- ✓ Estudio y análisis de los manejadores implementados para el SCADA “Guardián del ALBA”, y así aprovechar la reutilización de patrones y decisiones de diseño puestas en práctica en las soluciones mencionadas con anterioridad.
  - Estudio de la Interfaz Genérica de los Manejadores.
  - Estudio de las posibles bibliotecas, frameworks u ORM (Object Relational Mapping) para la comunicación con Sistemas de Bases de Datos.
  - Análisis y selección del componente de acceso a datos más adecuado.
  - Análisis y definición de los requisitos funcionales y no funcionales a partir de las características del manejador.
- ✓ Implementación y pruebas del manejador.

Después de obtener toda la información relacionada con el tema del trabajo se plantea como **posible resultado**: Un manejador para la comunicación con Sistemas de Bases de Datos para su integración al módulo de adquisición de datos del SCADA “Guardián del ALBA”.

Métodos Científicos Usados:

Teóricos:

- ✓ **Analítico Sintético:** Permitió el análisis de todo el contenido de características específicas del manejador y a su vez llegar a conceptos y generalizaciones.

- ✓ **Análisis-Histórico-Lógico**: Este método permitió el conocimiento de las diferentes herramientas y tecnologías necesarias para desarrollar el manejador destacando sus ventajas y desventajas.
- ✓ **Inductivo-Deductivo**: Permitió apreciar la necesidad existente en el proyecto SCADA "Guardián del ALBA", además de tener presente, las ventajas que ofrecen los manejadores hoy en día.

Empíricos:

- ✓ **Modelación**: Permitió la elaboración de los diferentes diagramas necesarios para el entendimiento y mantenimiento del producto.

El presente trabajo de Diploma está estructurado de la siguiente manera: Introducción, Cuatro Capítulos de Contenido, Conclusiones, Recomendaciones, Bibliografía, Anexos, y Glosario de Términos.

A continuación se hace una breve descripción del contenido de los capítulos.

- **Capítulo 1:** "Fundamentación Teórica", en este Capítulo se analiza con mayor profundidad el objeto de estudio. Se explica en detalles las metodologías y herramientas de desarrollo; además de las tecnologías actuales a considerar.
- **Capítulo 2:** "Características del Sistema", en este Capítulo se presenta las características que tendrá el manejador, así como una breve descripción de la solución propuesta.
- **Capítulo 3:** "Análisis y Diseño del Sistema", en este Capítulo se describe la arquitectura que se emplea en el desarrollo del manejador. También muestra los diagramas de clases de diseño, y las descripciones de las clases que forman parte de este diagrama de clases del diseño.
- **Capítulo 4:** "Implementación y Prueba del Sistema", en este Capítulo se muestran los diagramas de despliegue, de componentes, pruebas realizadas al manejador y se obtiene además una versión del producto final.

CAPÍTULO 1: "FUNDAMENTACIÓN TEÓRICA"

1.1 Introducción

En este capítulo se describen los sistemas SCADA y se explican las principales características de los manejadores en estos sistemas, además se realiza un análisis de las tendencias y tecnologías que son más utilizadas actualmente en el desarrollo de los manejadores.

1.2 Sistemas SCADA.

Los sistemas SCADA son aplicaciones de software para la adquisición, supervisión y control de datos mediante la comunicación con los dispositivos de campo, con la finalidad de controlar procesos automatizados desde la pantalla del ordenador. Estos sistemas automatizados están formados por diferentes programas de software que corren en una computadora y cuyo objetivo es visualizar todos los datos que se miden y que permita el control de los mismos de manera simple y efectiva. Provee información del proceso a diversos usuarios como son: operadores, supervisores de control de calidad, mantenimiento. Los sistemas SCADA interactúan con ordenadores, unidades remotas, sistemas de comunicación que efectúan las tareas de supervisión, gestión de los datos y el control total de los procesos. **(Guerra Garayta, 2009)**

1.2.1 Funciones Principales

Para comprender un poco más el funcionamiento de los sistemas SCADA resultaría de interés presentar las funcionalidades principales.

Un sistema SCADA debe cumplir con 3 funcionalidades principales:

- ✓ **Adquisición de datos:** consiste en recoger, procesar y almacenar la información recibida.
- ✓ **Supervisión:** permite observar desde un monitor la evolución de las variables de control.
- ✓ **Control:** permite modificar la evolución del proceso.

1.2.2 Manejadores de dispositivos en el Sistema SCADA "Guardián del ALBA"

En el sistema SCADA "Guardián del ALBA" el primer eslabón es la adquisición de datos, lo que permite obtener la información desde varios equipos como pueden ser los controladores lógicos programables (PLC), autómatas, sensores inteligentes, etc. Para obtener la información recogida por estos disímiles equipos se utilizan los controladores o manejadores de dispositivos (en inglés, *driver*), que se definen

como un software informático que le permite al sistema operativo interactuar con un periférico. En la solución SCADA "Guardián del ALBA" los manejadores son bibliotecas o componentes de software que permiten acceder a dispositivos y fuentes de datos a través de protocolos y estándares de comunicación, por lo que a su vez estos ejecutan dos tareas fundamentales que son: leer y escribir variables de los dispositivos de campos. **(García Hernández, 2009)**

En la actualidad es muy complicado tener en el software SCADA implementados todos los protocolos para toda variedad de dispositivos. A su vez lo común sería crear un protocolo genérico que cumpliera con gran parte de las especificaciones de cada uno de los protocolos y la tarea de interpretar esta genericidad se le encargaría a los manejadores. Es por ello que para este propósito surge la Interfaz Genérica de los Manejadores de Dispositivos (IGD) para el proyecto SCADA "Guardián del ALBA", que no es más que un conjunto de funciones definidas que brindan la posibilidad del acceso a los dispositivos industriales. Esta IGD presenta en su diseño una arquitectura compuesta por clases que representan características y funciones comunes de los manejadores desarrollados para el SCADA "Guardián del ALBA". **(Cedeño Pozo, 2009)**

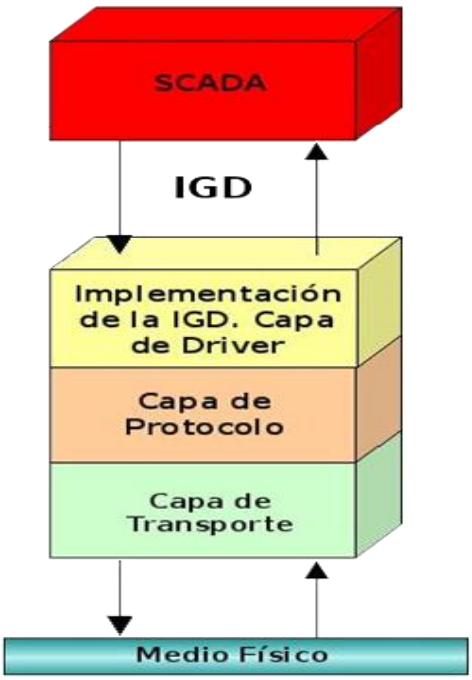

Figura 1: Interfaz Genérica de los Manejadores.

1.3 Componentes de Acceso a Datos

1.3.1 Bibliotecas de Acceso a Datos

- ✓ **LiteSQL**

LiteSQL es una biblioteca de acceso a la base de datos (BD) desarrollada en C++, que integra objetos fuertemente a la base de datos relacional y por lo tanto proporciona una capa de persistencia de objetos. LiteSQL soporta SQLite3, PostgreSQL y MySQL como *backend.* Además de la persistencia de objetos, LiteSQL establece relaciones orientadas a objetos como es el caso de la agregación, composición y asociación. Presenta características como:

1. Contiene operaciones relacionales (filtrar, ordenar, referencia a objetos de otro tipo).
2. Mantenimiento automático de la estructura de base de datos (crea y actualiza tablas).

- ✓ **SQLAPI**

SQLAPI es una biblioteca de acceso a la base de datos desarrollada en C++ que permite el acceso a múltiples bases de datos SQL como son: Oracle, SQL Server, MySQL, PostgreSQL. Utiliza las API para que las aplicaciones desarrolladas con SQLAPI puedan ejecutarse de manera rápida y eficiente. El producto también proporciona una interfaz de bajo nivel que le permite a los desarrolladores de bases de datos el acceso a características específicas. Presenta características como:

1. Presenta soporte libre, corrigiendo bugs y actualizaciones de la nueva versión.
2. Permite trabajar con un número de servidores de bases de datos SQL.
3. Proporciona mecanismos comunes de acceso a las bases de datos.

- ✓ **SOCI**

SOCI es una biblioteca de acceso a la base de datos desarrollada en C++ que hace la ilusión de la incrustación de consultas SQL en el código ordinario, quedando totalmente en el estándar de C++. La idea de esta biblioteca es proporcionarles a los programadores una manera de acceder a las bases de datos de la manera más natural e intuitiva. Presenta características como:

1. Soporta sistemas gestores de bases de datos como: Oracle, MySQL y PostgreSQL.

2. Permite la ejecución de consultas de forma dinámica sobre las bases de datos.
3. Su diseño está orientado a la elaboración de extensiones o *plug-in* que contribuyen a la incorporación y comunicación con nuevos Sistemas Gestores de Bases de Datos.
4. Reutilización: permite llamar a los métodos de un objeto de datos desde distintas partes de la aplicación.
5. Portabilidad: Utiliza una capa de abstracción que nos permite cambiar en mitad de un proyecto una base de datos MySQL a una Oracle sin ninguna complicación.
6. Mantenimiento del código: Gracias a la correcta ordenación de la capa de datos, las tareas de modificar y mantener el código se ejecutan de manera sencilla.

1.3.2 Mapeo de Objeto Relacional (ORM)

Los ORM constituyen una interface que une los modelos relacionales y el orientado a objetos. De forma general un ORM no es más que una técnica de programación que permite convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. Dentro de algunas ventajas de los ORM vale señalar que son capas que eliminan la incongruencia entre el mundo orientado a objetos y las bases de datos relacionales. Como principal desventaja de los ORM esta que estos se aplican a diseños de bases de datos relacionales que no deben variar en el tiempo por lo que más bien se aplican a diseños de bases de datos estáticos. Dentro de los ORM más usados en la actualidad están: Doctrine y NHibernate.

- ✓ **Doctrine**

Doctrine es un ORM que posee una poderosa capa de abstracción de BD. Una de sus características es la opción de escribir consultas de BD en un objeto apropiado orientado al dialecto SQL (Structured Query Language) y que se le denomina Lenguaje de Consulta de Doctrine (DQL), inspirado por el Lenguaje de Consulta de Hibernate (HQL). Este le proporciona a los desarrolladores una poderosa alternativa al SQL que mantiene la flexibilidad sin requerir duplicación de código innecesario. Una de las ventajas de utilizar esta capa de abstracción de objeto/relacional es que evita utilizar una sintaxis específica de un sistema de base de datos específico. Esta capa transforma de forma automática las llamadas a los objetos en consultas SQL optimizadas para el Sistema Gestor de Base de Datos que se

este utilizando en cada momento. De esta forma, es muy sencillo cambiar a otro sistema de base de datos completamente diferente en mitad del desarrollo de un proyecto. **(Bauta Camejo, 2009)**

- ✓ **NHibernate**

NHibernate es un framework de persistencia de objetos relacionales, que resuelve en forma automática la persistencia de objetos de dominio .NET. El objetivo principal de NHibernate es abstraer por completo al desarrollador de la base de datos asociada al proyecto en desarrollo, es decir, el desarrollador debe sólo trabajar con objetos, los cuales se pueden guardar en una base de datos utilizando métodos de los mismos objetos, pero nunca escribir ni analizar sentencias SQL en su código, siendo esta una de las principales funciones de un framework de persistencia. NHibernate describe los objetos o entidades persistentes y las relaciones mapeadas en XML, y genera automáticamente código SQL para la carga y almacenamiento de los objetos. **(Leyet Fernandez, 2008)**

1.3.3 Selección del Componente de Acceso a Datos

En este epígrafe se expusieron las principales características de los componentes de acceso a datos dentro de los cuales están los ORM y API de acceso a bases de datos. Los mismos son de gran uso a nivel internacional, puesto que presentan características que son provechosas para la programación de capas de acceso a los sistemas de bases de datos. En cuanto a la selección del componente adecuado se analizaron muchos aspectos. Para el desarrollo del manejador es una ventaja hacer uso de un componente de acceso a datos que contenga mecanismos que brinden la posibilidad de la comunicación con sistemas de bases de datos de forma dinámica y sin conocer la estructura de la misma. Por ello el estudio de estos componentes arrojó que en el análisis de los ORM aparece la principal desventaja y es que estos se aplican a diseños de bases de datos relacionales que no deben variar en el tiempo. Teniendo en cuenta que el componente debe tener características específicas se decide hacer uso de la biblioteca de acceso a datos **SOCI** teniendo como principales características que esta biblioteca permite la ejecución de consultas de forma dinámica sobre las bases de datos, y presenta un diseño orientado a la elaboración de extensiones o plug-in para la incorporación y comunicación con nuevos sistemas gestores de bases de datos**.**

1.4 Sistemas Gestores de Bases de Datos.

Un Sistema Gestor de Base de Datos (SGBD) es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Estos se componen de un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de consultas. Un SGBD permite definir los datos a distintos niveles de abstracción y a su vez manipular los mismos, garantizando su seguridad e integridad. Los gestores más usados hoy en día son: Oracle, MySQL, y PostgreSQL; los cuales cumplen con la política de ser multiplataforma.

Un SGBD debe permitir:

- ✓ Definir una base de datos: consiste en especificar tipos, estructuras y restricciones de datos.
- ✓ Construir la base de datos: consiste en guardar los datos en algún medio controlado por el mismo SGBD.
- ✓ Manipular la base de datos: permite realizar consultas, actualizarla, y generar informes.

1.4.1 MySQL

MySQL fue creado por la empresa sueca MySQL AB, que mantiene el copyright del código fuente del servidor SQL, así como también de la marca. Es un sistema de gestión de bases de datos relacional, licenciado bajo la GPL (Licencia Pública General) de la GNU. Su diseño multihilo le permite soportar una gran carga de datos de forma muy eficiente. Es un gestor de bases de datos sencillo de usar e increíblemente rápido. También es uno de los motores de bases de datos más usados en Internet, la principal razón de esto es que el mismo es gratis para aplicaciones no comerciales. Soporta grandes bases de datos y presenta ventajas como: facilidad de configuración e instalación, velocidad al realizar las operaciones; lo que le hace uno de los gestores con mejor rendimiento. Este gestor de bases de datos es muy potente, y es muy usado en la actualidad.

1.4.2 Oracle

Oracle es un sistema de gestión de bases de datos relacional, desarrollado por Oracle Corporation. El mismo es considerado uno de los gestores más completos en la actualidad por ofrecer soporte de transacciones, estabilidad, escalabilidad. Oracle es básicamente una herramienta cliente/servidor para la gestión de bases de datos. Es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hacen que sólo se vea en empresas muy grandes y multinacionales; por

norma general. El mismo tiene ventajas como: presenta un aceptable soporte, es la base de datos con más orientación hacía INTERNET y una de las ventajosas características que posee Oracle es el Patrón de Consulta. Esta poderosa herramienta lógica de SQL permite el reconocimiento de un patrón de consulta mediante su búsqueda por nombre, dirección u otro dato parcialmente recordado. Este patrón es muy importante a la hora de realizar consultas, puesto que es muy común que se necesite un texto y no se recuerde cómo fue ingresado el mismo.

1.4.3 PostgreSQL

PostgreSQL es un sistema de base de datos objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones e integridad transaccional. PostgreSQL es un poderoso sistema de bases de datos puesto que está diseñado para administrar grandes cantidades de datos, el mismo tiene la fama de ser la base de datos de código abierto (Open Source) más avanzada del mundo. La utilización de este robusto gestor de base de datos trae consigo muchas ventajas; por ejemplo: tiene buen soporte para triggers y procedimientos en el servidor, funciona en todos los sistemas operativos importantes incluyendo Linux, UNIX, y Windows. Entre sus principales características se encuentran:

- Soporta casi toda la sintaxis SQL pues tiene soporte total para llaves extranjeras, joins, vistas, disparadores y procedimientos almacenados (en múltiples lenguajes).

- Integridad transaccional: Obedece completamente a la especificación *ACID* (en español ***A**tomicidad, **C**onsistencia, **A**islamiento y **D**urabilidad*). Cliente/Servidor.

- Usa una arquitectura proceso-por-usuario cliente/servidor, para manejar procesos.

1.5 Metodología y Herramientas de Desarrollo

Para la realización de una aplicación informática se deben definir las metodologías y herramientas que serán de mayor utilidad para su implementación. A continuación se explicarán brevemente los detalles de cada una de estas metodologías y herramientas seleccionadas para el desarrollo del manejador.

1.5.1 Proceso Unificado de Desarrollo (RUP)

RUP es un proceso bien definido, estructurado y adaptable específicamente a cada proyecto. Está definido por tres características fundamentales: dirigido por los casos de uso, centrado en la arquitectura y ser iterativo e incremental.

En RUP los casos de uso guían su diseño, implementación y prueba, constituyendo así un elemento que guía el trabajo. Existe una relación entre los casos de uso y la arquitectura debido a que los casos de uso deben encajar en la arquitectura cuando se llevan a cabo y la arquitectura debe permitir el desarrollo de todos los casos de uso. Esto trae consigo que tanto la arquitectura como los casos de uso evolucionen conjuntamente durante todo el proceso de desarrollo de software.

RUP es un proceso iterativo e incremental que permite dividir el trabajo en partes pequeñas, logrando así un equilibrio entre la arquitectura y casos de uso durante cada proyecto. Durante las iteraciones del proyecto se obtiene un incremento, provocando un aumento del producto.

RUP está definido por 4 fases fundamentales:

- ✓ **Inicio:** el objetivo es determinar la visión del proyecto.
- ✓ **Elaboración:** el objetivo es determinar la arquitectura óptima.
- ✓ **Construcción:** el objetivo es llegar a obtener la capacidad operacional inicial.
- ✓ **Transición:** el objetivo es llegar a obtener la liberación del proyecto.

1.5.2 Lenguaje Unificado de Modelado (UML).

UML utiliza herramientas que permiten el modelado visual lo cual facilita la gestión de dichos modelos, lo que posibilita ocultar o exponer detalles cuando sea necesario. Esto ayuda a mantener la consistencia entre los artefactos del sistema: requisitos, diseños e implementaciones. UML se ha convertido en un lenguaje estándar para representar y modelar la información con la que se trabaja en las fases de análisis y diseño. Este lenguaje constituye un método formal de modelado aportando como ventajas: mayor rigor en la especificación, pues posibilita realizar una verificación y validación del modelo realizado, además de automatizar determinados procesos. UML como lenguaje de modelado ayuda a la especificación, visualización, construcción y documentación de los artefactos de un proceso de sistema intensivo.

1.5.3 Herramientas CASE (Ingeniería de Software Asistida por Ordenador)

- ✓ **Rational Rose**

Rational Rose es una herramienta CASE para el diseño de software. La misma esta destinada al modelado visual y construcción de componentes de aplicaciones de software. Permite establecer la trazabilidad entre diferentes diagramas. Rational Rose se integra con otras herramientas en su ambiente para soportar el trabajo colaborativo. Permite especificar, analizar, y diseñar el sistema antes de codificarlo. Propone la utilización de cuatro tipos de modelos para realizar el diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, una lógica y otra física. Rational Rose utiliza un proceso de desarrollo iterativo controlado, donde el desarrollo se lleva a cabo en una secuencia de iteraciones. Se puede generar código en distintos lenguajes de programación a partir de un diseño en UML.

- ✓ **Visual Paradigm**

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas y a su vez generar documentación. Esta herramienta multiplataforma está diseñada para que los usuarios realicen la construcción de sistemas de software a gran escala de forma fiable a través de la utilización de la aproximación orientada a objetos. Visual Paradigm es un producto que facilita a las organizaciones diseñar visualmente y con diagramas, integrar y desplegar sus aplicaciones empresariales y sus respectivas bases de datos. La herramienta ayuda al equipo de desarrollo de software a destacarse en el proceso de modelado-construcción-despliegue, maximizando y acelerando las contribuciones del equipo y los individuos.

Aspectos	Visual Paradigm	Rational Rose
Modelado de BD	Si	Si
Metodología de Desarrollo	Independiente de la Metodología	Orientado a RUP
Integración con el IDE	Alto	Bajo
Generación de Código	Alto	Bajo

Generación de Doc.	Si	Si
Soporte	Multiplataforma	Multiplataforma
Modelado UML	Si UML 2.1	Si UML 1.0

Tabla 1: Comparación entre Herramientas Case.

1.5.4 Entorno Integrado de Desarrollo (IDE)

- ✓ **Eclipse**

Eclipse se define como una plataforma, abierta para todo y para nada en particular. Está diseñado para que sea extensible indefinidamente con la adecuada implementación de *plug-in*. Es un IDE porque provee de herramientas que facilitan el trabajo en el desarrollo de software, permite administrar el espacio de trabajo (en inglés *workspace*), y además compilar, correr y depurar aplicaciones, posee herramientas que permiten compartir elementos y control de versión sobre el código fuente. Este IDE es una gran estructura formada por un núcleo y muchos plug-ins que van conformando la funcionalidad final. El mismo tiene una gran capacidad y versatilidad, lo cual es permitido por su arquitectura.

- ✓ **Visual Studio**

Microsoft Visual Studio es un Entorno Integrado de Desarrollo para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

- ✓ **KDevelop**

KDevelop es un Entorno Integrado de Desarrollo para sistemas GNU/Linux y otros sistemas Unix, publicado bajo la licencia GPL, el mismo está orientado principalmente al uso bajo el entorno gráfico KDE. Tiene una interfaz de desarrollo la cual cuenta con un compilador, por lo que depende de gcc para producir código binario.

La última versión de este IDE se encuentra actualmente bajo desarrollo y funciona con distintos lenguajes de programación como C, C++, Java, SQL, Python, Pascal, etc.

1.5.5 Lenguaje de Programación.

- ✓ **C++**

El lenguaje de programación C++ fue creado en la década de 1980 por Bjarne Stroustrup como extensión del lenguaje C. Este lenguaje es utilizado ampliamente en la industria del software. En este caso se cuenta con varias razones para seleccionarlo como lenguaje base de la solución propuesta. El conocimiento con que cuenta el equipo de desarrollo acerca de este lenguaje y el hecho de que la concepción del SCADA "Guardián del ALBA" desde su definición ha sido el desarrollo de soluciones integrales hechas en C++ influyeron en gran medida para su selección, además los recursos con que cuenta son muy útiles en la rama de la automatización, que es la base de este producto. Características generales:

- ✓ Es un lenguaje de programación híbrido.
- ✓ Es un lenguaje multiplataforma, orientado a objetos e imperativo.
- ✓ Usa tipos de datos fuertes y estáticos.
- ✓ Manejo de memoria por parte del programador, lo que permite un mejor control de esta y una buena administración de recursos de computadora.
- ✓ C++ permite trabajar tanto a alto como a bajo nivel.

1.5.6 Selección de Metodología y Herramientas.

Para el desarrollo del manejador se debe seleccionar un Entorno Integrado de Desarrollo que permita la programación en C++, y a su vez facilite la vinculación de bibliotecas por lo que se escogió el IDE Eclipse. Como Herramienta Case se seleccionó Visual Paradigm, por las principales características que presenta esta herramienta. Todo el proceso estará guiado por la metodología RUP, a su vez estará representado por el Lenguaje UML y además se usará como lenguaje de programación C++.

Conclusiones

En este capítulo se abordaron conceptos importantes para entender el funcionamiento de un sistema SCADA. Se analizaron las metodologías, herramientas y tecnologías más utilizadas a nivel mundial. Dentro de ellas fueron seleccionadas las más propicias para el desarrollo del manejador, teniendo en

cuenta sus características y su aplicación en las nuevas tendencias actuales. Se seleccionó como componente de acceso a datos la biblioteca SOCI por tener características beneficiosas para el desarrollo del manejador. En el desarrollo de la ingeniería de software se decidió utilizar la metodología RUP, el lenguaje de modelado UML, como Herramienta CASE a Visual Paradigm, y el lenguaje de programación escogido para desarrollar la solución es C++.

CAPÍTULO 2: "CARACTERÍSTICAS DEL SISTEMA"

2.1 Introducción

Este capítulo refleja una descripción de la solución propuesta para resolver el problema científico planteado. Se expone el diagrama del modelo de dominio correspondiente al manejador para lograr un mejor entendimiento del contexto en que se utilizará la solución y además se detallan los aspectos que conforman dicho modelo de dominio. Además se especifican los requisitos tanto funcionales como no funcionales.

2.2 Protocolo de Comunicación SQL

Un protocolo es una convención o estándar que controla o permite la comunicación y transferencia de datos. En su forma más simple, un protocolo de comunicación puede ser definido como las reglas que dominan la sintaxis, semántica y sincronización de la comunicación. El Lenguaje SQL (Lenguaje de Consulta Estructurado) no es más que un lenguaje estándar que permite la comunicación con sistemas de bases de datos. Por tanto, es un lenguaje normalizado que permite la combinación de los distintos tipos de lenguajes de alto nivel con cualquier gestor de base de datos. El beneficio del cliente se centra a que el manejador realice solamente operaciones de lectura y escritura. Esto suprime en la utilización a los grupos de comandos que no cumplan con este objetivo.

En vista de que este lenguaje estándar brinda varios comandos, se realizó un estudio sobre las características específicas de cada uno de ellos. El mismo arrojó como resultado, que para obtener un uso eficiente de las capacidades del protocolo en cuanto a las tareas de lectura y escritura de variables, se uso la sentencia *SELECT* para el caso de las tareas de lectura y para las tareas de escritura la sentencia *UPDATE*.

De manera general la mayoría de las sentencias SQL tienen la misma estructura. Todas comienzan por un verbo (*SELECT*), a continuación le sigue una o más cláusulas que nos dicen los datos con los que vamos a operar *(FROM, WHERE)*, algunas de estas son opcionales y otras obligatorias como es el caso del *FROM*, luego las condiciones más especificas que pueden ser opcionales en dependencia de lo que se quiera devolver *(INNER JOIN)*.

2.3 Direccionamiento de Variables

2.3.1 Variables SQL

En el SCADA "Guardián del ALBA" se manejan conceptos generales del término *variable*. En el desarrollo del manejador este concepto se trata como una variable dentro de una base de datos. Una base de datos es un conjunto de datos interrelacionados, no redundantes y persistentes, con una estructura claramente definida basada en un modelo de datos y almacenados en un soporte informático, y a su vez organizado de forma independiente de su utilización y accesible simultáneamente por distintos usuarios y aplicaciones**. (Contreras Rodríguez, 2009)**.

Teniendo en cuenta las características del manejador, una variable SQL se define como una o varias *tuplas,* devueltas por una consulta SQL sobre un campo de una tabla de la base de datos.

Características:

- ✓ El tipo de dato del campo en la tabla solo puede ser de tipo cadena de caracteres, numérico o booleano.
- ✓ Se tomará como valor de la variable solo el primer registro devuelto por la consulta SQL que representa la dirección de una variable.

2.3.2 Formato de dirección de las Variables SQL

Los manejadores de forma general para establecer una comunicación deben estar regidos por las reglas de un protocolo. A su vez para este protocolo se define un formato de dirección de las variables, en donde se define los parámetros que deben ser especificados y de esta forma quedar descritas las reglas que permitirán la comunicación.

Parámetros de dirección:

--t *valor* **--c** *valor* **--u** *valor* **--w** *valor*

Donde:

t: El parámetro "--t" es para especificar la tabla de donde se va a leer el campo (o variable) y cuyo valor es una cadena de caracteres.

c: El parámetro "--c" es para especificar el nombre del campo (o variable) como salida que se va a leer en la tabla anteriormente especificada.

u: El parámetro "--u" es para especificar las uniones entre varias tablas; se hace uso de este parámetro con el objetivo de realizar una consulta un poco más compleja. El formato es: el nombre de la tabla seguido del carácter "." seguido del nombre del campo, seguido del tipo de dato del campo (**i** para los números enteros, **s** para las cadenas de caracteres y **b** para los booleanos) y se compara mediante el signo de igualdad o carácter "=" con la estructura anterior pero de otra tabla. Para separar una unión de otra se utiliza el carácter "," como forma de un AND lógico.

w: El parámetro "--w" se para especificar las condiciones para la realización de la consulta, para esto se debe tener en cuenta:

a) Se pueden agrupar condiciones encerradas entre paréntesis.

El carácter ","hace función de un AND lógico y el carácter "|" de un OR.

El formato de este parámetro es: el nombre de la tabla seguido del carácter "." seguido del nombre del campo, seguido del tipo de dato del campo (**i** para los números enteros, **s** para las cadenas de caracteres y **b** para los booleanos), luego se le asigna un valor constante a esa condición, que puede ser un valor numérico, una expresión o una cadena de caracteres. En este parámetro se hará uso de los operadores > y < para establecer comparaciones, así como del operador = para las asignaciones de valores.

Ejemplo de dirección completa:
--t nombre_tabla --c nombre_del_campo: s --u nombre_tabla2.campo1: i=nombre_tabla.campo1: i, nombre_tabla3.campo2: b=nombre_tabla.campo2: b --w (nombre_tabla2.campo3: s=cuba, nombre_tabla.campo4: i>40) | nombre_tabla1.campo3: i >0.33 | nombre_tabla1.campo3: i=0.1

2.4 Descripción de la Solución

El manejador que se propone desarrollar esta basado en la comunicación con Sistemas Gestores de Bases de Datos, siendo esto una gran ventaja para el SCADA “Guardián del ALBA”, puesto que brinda la posibilidad de ampliar la adquisición de datos desde otras fuentes de información. La interacción entre el manejador y los Sistemas Gestores de Bases de Datos y a su vez con el Sistema SCADA, así como el Sistema de Terceros con los Sistemas Gestores de Bases de Datos se efectuará mediante el envío y recepción de mensajes SQL, que estarán conformados por consultas SQL, lo cual permitirá la

ejecución de las funciones de lectura y escritura de variables mediante las sentencias *SELECT* y *UPDATE*.

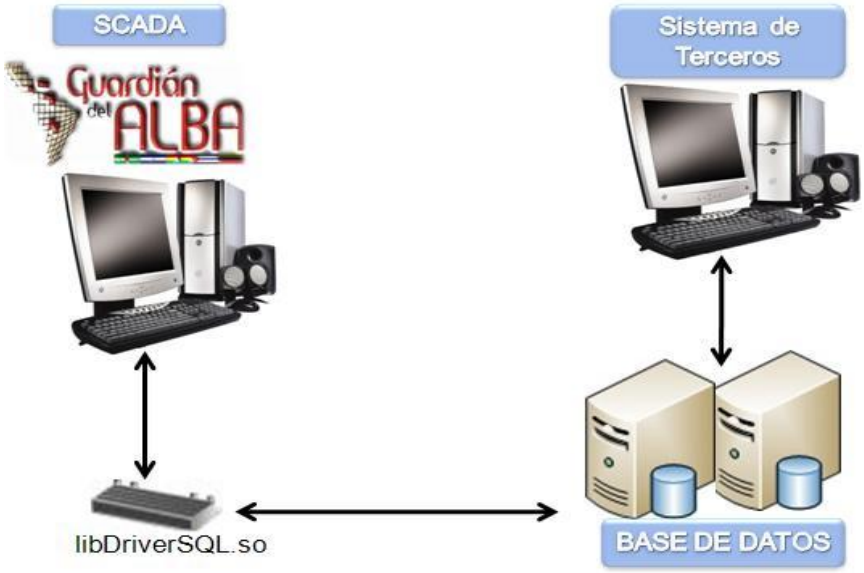

Figura 2: Propuesta de solución del DriverSQL.

2.5 Modelo de Dominio

El modelo de dominio es una representación visual de los objetos más significativos en un problema determinado. Se realiza una vez que se ausenta una estructura definida para los procesos del negocio.

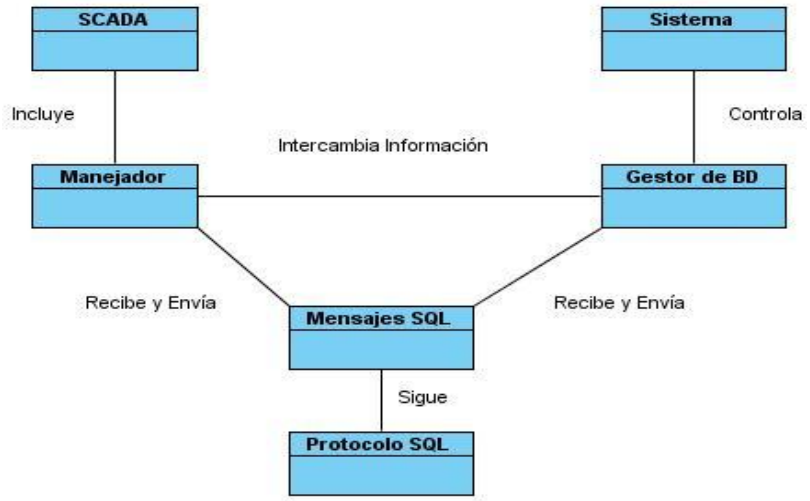

Figura 3: Modelo de Dominio.

2.5.1 Análisis de los conceptos del modelo de dominio.

Se define:

SCADA: En este concepto se engloban todos los componentes que conforman al Sistema de Adquisición, Supervisión y Control de Datos desarrollado en el CEDIN.

Manejador: Módulo del SCADA "Guardián del ALBA" que se encarga del intercambio de información con fuentes de datos.

Mensajes SQL: Representa cualquier tipo de datos que se pueda enviar o recibir desde el subsistema de recolección hasta la Base de Datos.

Protocolo SQL: Conjunto de reglas o leyes que rigen la comunicación entre el manejador y la base de datos.

Gestor de BD: Fuente de datos para la adquisición de información la cual se comunica con el manejador mediante mensajes SQL.

Sistema: Cualquier sistema de terceros que interactúa con la Base de Datos.

2.6 Especificación de Requisitos

Para los manejadores desarrollados para el SCADA "Guardián del ALBA", existe un grupo de requisitos; funcionales y no funcionales, definidos previamente. Estos requisitos están especificados y descritos en el documento: "Especificación de Requerimientos de Software del Módulo de Drivers", versión 1.3. **(Trujillo, 2006).** De los requisitos que se especifican en el documento antes mencionado, el manejador que se propone desarrollar debe incluir en su funcionamiento, específicamente los siguientes requisitos funcionales:

2.6.1 Requisitos Funcionales

1. Definir parámetros de conexión y dirección de consulta.
2. Configurar manejadores, dispositivos y transporte.
3. Validar direcciones de consultas.

4. Ejecutar consultas de lecturas de datos.
5. Ejecutar consultas de escrituras de datos.
6. Establecer estampa de tiempo.
7. Brindar información de diagnóstico.
8. Brindar mensajes de error.

2.6.2 Requisitos no Funcionales

En el caso de los requerimientos no funcionales se tendrán en cuenta para este manejador todos los que se mencionan en el documento de especificación de requerimientos antes mencionado. **(Trujillo, 2006).**

CAPÍTULO 3: "ANÁLISIS Y DISEÑO DEL SISTEMA"

3.1 Introducción

En este capítulo se expondrán aspectos que tributan al análisis y diseño del manejador. Se define la arquitectura, el diagrama de clases del diseño, la descripción de estas clases, las cuales a su vez estarán presentes durante todo el desarrollo.

3.2 Interfaz Genérica de los Manejadores

Actualmente en el mundo existen una gran variedad de protocolos industriales que rigen la comunicación con los dispositivos. Un protocolo es distinto de otro en cuanto a restricciones de lectura y escritura de información, así como en términos de configuración se refiere. Dadas estas condiciones puede resultar costoso realizar cambios para que se adquiera una adaptación a diferentes protocolos. Es por ello que para solucionar la mayoría de las diferencias entre un protocolo y otro se diseñó la Interfaz Genérica de los Manejadores de Dispositivos, la cual constituye una especificación de un conjunto de características y funcionalidades más comunes de los manejadores de dispositivos. Esta Interfaz provee de una arquitectura que garantiza que el desarrollo de los manejadores sea un poco más eficiente en cuanto a tiempo y recursos utilizados por parte del sistema operativo. La misma esta compuesta por la biblioteca DriverCore, la que ofrece un conjunto de clases que en la mayoría de los casos permite el desarrollo de nuevos drivers. Esta biblioteca a su vez permite cargar opcionalmente una biblioteca de transporte asíncrona (basada en ASIO 1.4.1) que contiene la definición de tipos de transportes TCP y Serial.

3.3 Arquitectura de Software (AS)

La AS es la organización fundamental de cualquier sistema representada en sus componentes, las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución. Su objetivo principal es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permita la comunicación entre los equipos que participen en un proyecto. **(Pimienta Fernández, 2009)**

3.3.1 Patrones de Arquitectura

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. **(Ruz Pérez, 2009)**

3.3.1.1 Arquitecturas en Capas

La arquitectura en capas constituye una especialización de la arquitectura cliente-servidor, donde la carga se distribuye en partes con un reparto claro de las funciones y donde cada capa tiene relación con la siguiente. Este patrón simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo así las dependencias de forma que las capas más bajas no son conscientes de ningún detalle de las capas superiores. **(Páez Castillo, 2009)**

Principales estilos del patrón en capas.

- ✓ Arquitecturas de dos capas.
- ✓ Arquitecturas de tres capas.
- ✓ Arquitecturas de n capas.

En el diseño del manejador se utiliza el estilo de arquitectura de tres capas específicamente, ya que el modelo de diseño del mismo está organizado en las capas de Driver, Protocolo y Transporte, como se muestra a continuación:

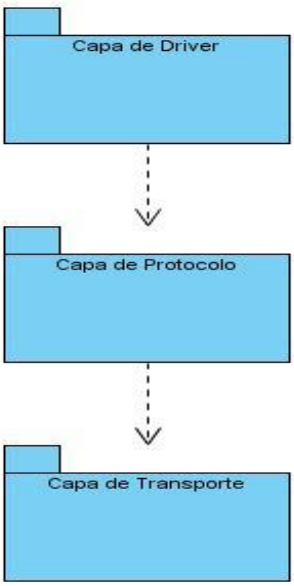

Figura 4: Modelo de Diseño de Capas del DriverSQL.

- ✓ **Capa de Driver**

En la capa de Driver se definen e implementan las clases que se relacionan con el DriverCore. Primeramente es necesario implementar las metaclases, las que permiten que los manejadores tengan introspección. Para que esto se logré se hace necesario que las clases que se vayan a implementar hereden de las clases Device, Driver, DriverMetaClass y DeviceMetaClass que son brindadas por la Interfaz Genérica de los Manejadores. Las clases Device y Driver modelan características comunes de los manejadores y los dispositivos respectivamente. Otras clases que también deben ser implementadas son las clases Block y la IProtocolAddress. En el caso de la clase IProtocolAddress es la encargada de la validación de las direcciones a partir de las características específicas del protocolo y para ello se crea una clase que sea descendiente directa de ella. Por su parte en la clase Block se construyen dinámicamente los bloques creados por los manejadores cada vez que se asocian a los dispositivos. De forma general estas clases representan características específicas de los manejadores y además garantizan que la información recogida cumpla con la Interfaz Genérica de los Manejadores.

- ✓ **Capa de Protocolo**

El objetivo de la capa de Protocolo es garantizar la lógica de comunicación entre el manejador y las bases de datos y para ello se define la clase propia del protocolo que hereda de la clase EndPoint; la cual implementa funcionalidades que permiten el intercambio de información con las bases de datos. Además en esta clase se controlan todos los pasos que se definen por el protocolo para lograr una buena comunicación, y que esta a su vez sea segura y esté preparada para responder a los fallos que pudieran surgir. Se utiliza también un transporte el cual permite, a través del medio físico; el intercambio de información.

- ✓ **Capa de Transporte**

La capa de Transporte contiene las clases de enlace con la interfaz que proporciona la biblioteca de acceso a datos SOCI**,** la cual tiene como objetivo establecer la comunicación con las bases de datos a través de la red. También se utiliza definiciones de la biblioteca de transporte TransportProvider, la que brinda la posibilidad de lectura y escritura de datos a través del medio físico de manera asíncrona.

3.4 Diagrama de Clases del Diseño

El siguiente diagrama representa el diseño de las clases más importantes durante el desarrollo del manejador.

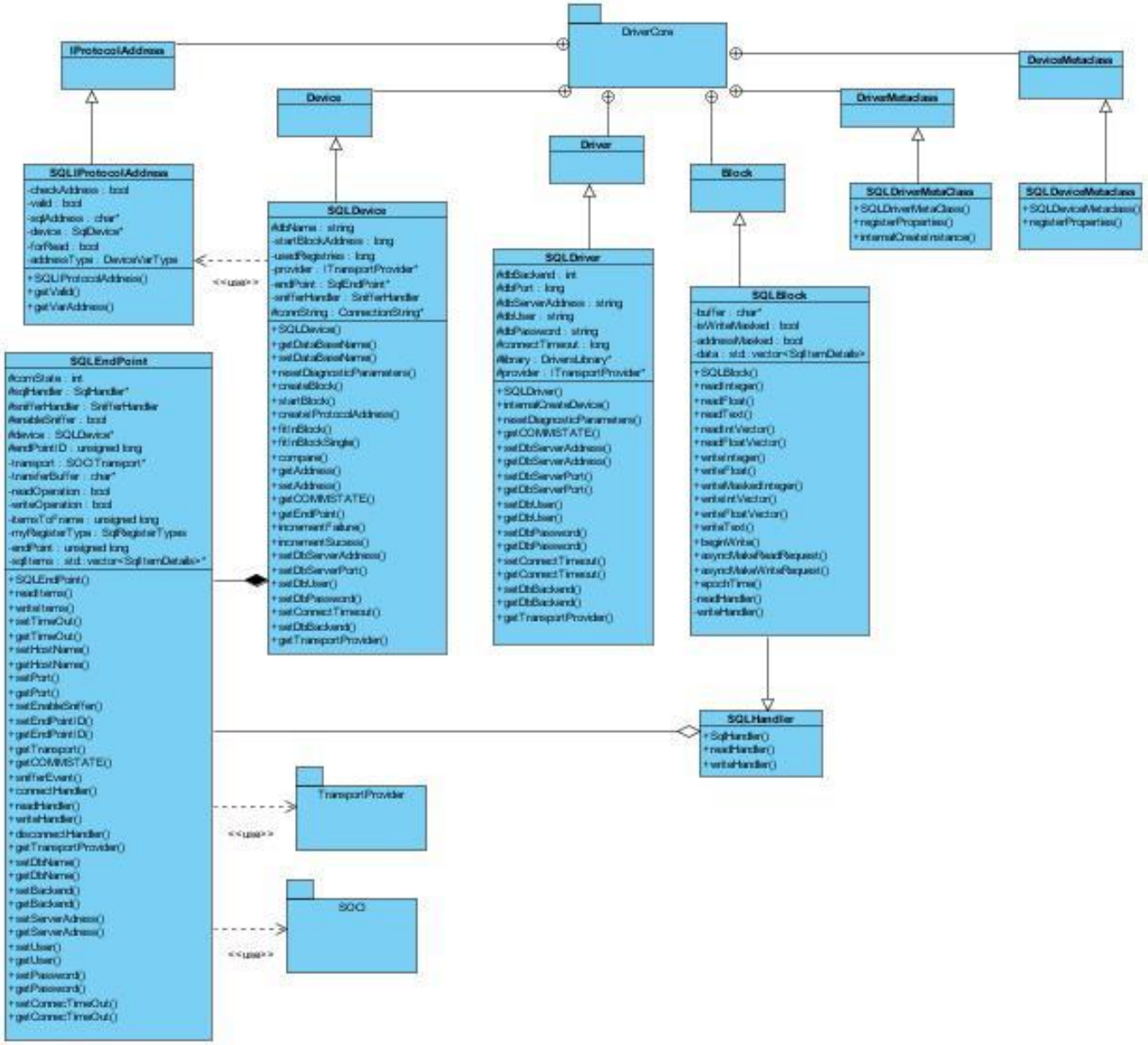

Figura 5: Diagrama de Clases del Diseño del DriverSQL.

3.5 Descripción de las Clases del Diseño

Clase del Diseño SqlEndPoint.

Nombre: SqlEndPoint
Tipo de clase:

Atributo	Tipo
sqlHandler	SqlHandler *
endPointID	unsigned long
device	SqlDevice *
snifferHandler	SnifferHandler
enableSniffer	bool
myRegisterType	SqlRegisterTypes
transport	SOCITransport*
transferBuffer	char*
readOperation	bool
writeOperation	bool
sqlItems	vector <SqlItemDetails>*
comState	int

Para cada responsabilidad:	
Nombre:	SqlEndPoint()
Descripción:	Constructor por Defecto.
Nombre:	readItems()
Descripción:	Función que permite leer un conjunto de valores a partir de los nombres de los elementos.
Nombre:	writeItems()
Descripción:	Función que permite escribir un conjunto de valores a partir de los nombres de los elemento.
Nombre:	setTimeOut()
Descripción:	Función para establecer el tiempo de espera para la recepción del mensaje.

Nombre:	getTimeOut()
Descripción:	Función que devuelve el tiempo de espera para recibir el mensaje.
Nombre:	setHostName()
Descripción:	Función para establecer la dirección del host.
Nombre:	getHostName()
Descripción:	Función que devuelve la dirección del host.
Nombre:	setPort()
Descripción:	Función que permite establecer el puerto.
Nombre:	getPort()
Descripción:	Función que devuelve el puerto.
Nombre:	setDbName()
Descripción:	Función para establecer el nombre de la BD.
Nombre:	getDbName()
Descripción:	Función que devuelve el nombre de la BD.
Nombre:	setBackend()
Descripción:	Función para establecer el backend de la BD.
Nombre:	getBackend()
Descripción:	Función que devuelve el backend de la BD.
Nombre:	setServerAdress()
Descripción:	Función para establecer el ServerAdress.
Nombre:	getServerAdress()
Descripción:	Función que devuelve el ServerAdress.
Nombre:	setUser()
Descripción:	Función para establecer el usuario para conexión a la BD.

Nombre:	getUser()
Descripción:	Función que devuelve el usuario para lograr la conexión a la BD.
Nombre:	setPassword()
Descripción:	Función para establecer la contraseña para conexión a la BD.
Nombre:	getPassword()
Descripción:	Función que devuelve la contraseña para lograr la conexión a la BD.
Nombre:	setConnectTimeout()
Descripción:	Función para establecer el tiempo de espera.
Nombre:	getConnectTimeout()
Descripción:	Función que devuelve el tiempo de espera.
Nombre:	setEnableSniffer()
Descripción:	Función que establece el uso o no del Sniffer.
Nombre:	setEndPointID()
Descripción:	Función que cambia el identificador del EndPoint.
Nombre:	getEndPointID()
Descripción:	Función que devuelve el identificador del EndPoint.
Nombre:	getTransport()
Descripción:	Función que devuelve una instancia del transporte.
Nombre:	getCOMMSTATE()
Descripción:	Función que devuelve el estado de la comunicación con el dispositivo.
Nombre:	snifferEvent()
Descripción:	Evento del Sniffer que se lanza en las operaciones realizadas en la transacción.
Nombre:	readHandler()
Descripción:	Handler de lectura del transporte reimplementado.

Nombre:	writeHandler()
Descripción:	Handler de escritura del transporte reimplementado.
Nombre:	connectHandler()
Descripción:	Handler de conexión del transporte reimplementado.
Nombre:	disconnectHandler()
Descripción:	Handler de desconexión del transporte reimplementado.
Nombre:	getTransportProvider()
Descripción:	Función que devuelve una instancia del TransportProvider.

Tabla 2: Clase del Diseño SqlEndPoint.

Clase del Diseño SqlDriver.

Nombre: SqlDriver	
Tipo de clase:	
Atributo	Tipo
library	DriversLibrary *
provider	ITransportProvider*
dbBackend	int
dbPort	long long
connectTimeout	long long
dbServerAdress	string
dbUser	string
dbPassword	string
Para cada responsabilidad:	

Nombre:	SqlDriver()
Descripción:	Constructor por Defecto.
Nombre:	internalCreateDevice()
Descripción:	La función responde por la creación de una instancia de SqlDevice. Es llamada cada vez que se requiere la creación de un dispositivo Sql.
Nombre:	resetDiagnosticParameters()
Descripción:	Restablece los parámetros de diagnóstico del manejador.
Nombre:	getCOMMSTATE()
Descripción:	La función getCOMMSTATE() es un método que devuelve el valor del estado del dispositivo Sql.
Nombre:	setDbServerAdress()
Descripción:	Función que establece el valor de dbServerAdress.
Nombre:	getDbServerAdress()
Descripción:	Función que devuelve el valor de dbServerAdress.
Nombre:	setDbServerPort()
Descripción:	Función que establece el puerto de conexión a la BD.
Nombre:	getDbServerPort()
Descripción:	Función que devuelve el puerto de conexión a la BD.
Nombre:	setDbUser()
Descripción:	Función que establece el usuario de conexión a la BD.
Nombre:	getDbUser()
Descripción:	Función que devuelve el usuario de conexión a la BD.
Nombre:	setDbPassword()
Descripción:	Función que establece la contraseña de conexión a la BD.
Nombre:	getDbPassword()

Descripción:	Función que devuelve la contraseña de conexión a la BD.
Nombre:	setConnectTimeout()
Descripción:	Función que establece el tiempo de espera.
Nombre:	getConnectTimeout()
Descripción:	Función que devuelve el tiempo de espera.
Nombre:	setDbBackend()
Descripción:	Función que establece el backend de la BD.
Nombre:	getDbBackend()
Descripción:	Función que devuelve el backend de la BD.
Nombre:	getTransportProvider()
Descripción:	Función que devuelve una instancia del TransportProvider

Tabla 3: Clase del Diseño SqlDriver.

Clase del Diseño SqlIProtocolAddress.

Nombre: SqlIProtocolAddress	
Tipo de clase:	
Atributo	Tipo
device	SqlDevice*
forRead	bool
varAddress	string
addressType	DeviceVarType
Para cada responsabilidad:	
Nombre:	SqlIProtocolAddress()
Descripción:	Constructor por Defecto.
Nombre:	getValid()

Descripción:	Función que retorna un valor que expresa si la dirección es válida o no de acuerdo a las reglas específicas del protocolo.
Nombre:	getVarAddress()
Descripción:	Función que retorna la dirección de la variable.

Tabla 4: Clase del Diseño SqlIProtocolAddress.

Clase del Diseño SqlHandler.

Nombre: SqlHandler	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	SqlHandler()
Descripción:	Constructor por Defecto.
Nombre:	readHandler()
Descripción:	Función que se ejecuta cuando se termina satisfactoriamente o no la comunicación con el dispositivo Sql a través de un comando de lectura. Este método solo se dispara cuando la transacción ha llegado a su fin.
Nombre:	writeHandler()
Descripción:	Función que se ejecuta cuando se termina satisfactoriamente o no la comunicación con el dispositivo Sql a través de un comando de escritura. Este método solo se dispara cuando la transacción ha llegado a su fin.

Tabla 5: Clase del Diseño SqlHandler.

Las descripciones de las clases restantes pueden ser consultadas en los anexos.

CAPÍTULO 4: "IMPLEMENTACIÓN Y PRUEBA DEL SISTEMA"

4.1 Introducción

En este capítulo se tratan temas que contribuyen a la implementación y pruebas del manejador. Se exponen algunos artefactos que son productos del resultado de las actividades de estos flujos de trabajo como es el caso del grupo de Diagramas, y Pruebas que forman parte de estos artefactos generados.

4.2 Estándar de Codificación

El código de los manejadores desarrollados para el SCADA "Guardián del ALBA" sigue estándares predefinidos para la línea de manejadores. Su programación es en idioma inglés debido a que gran parte de las palabras son simples, no se acentúan y a su vez es un idioma muy circulado en el mundo informático.

4.2.1 Nombre de los Ficheros

Los ficheros tendrán la siguiente estructura:

- ✓ NameOfUnits.h
- ✓ NameOfUnits.cpp

4.2.2 Clases

Los nombres de las clases deben tener la siguiente forma: MyClass. Los ficheros .h solo definirán una clase y a dicha definición de clase le debe corresponder un fichero.cpp donde se implementen las funcionalidades de la clase definida.

4.2.3 Métodos

Los métodos de las clases deben seguir la siguiente estructura: myFunction; excepto los constructores y destructores, todos los métodos deben empezar con minúscula.

4.3 Diagrama de Despliegue

.

Figura 6: Diagrama de Despliegue del DriverSQL.

4.4 Diagrama de Componentes

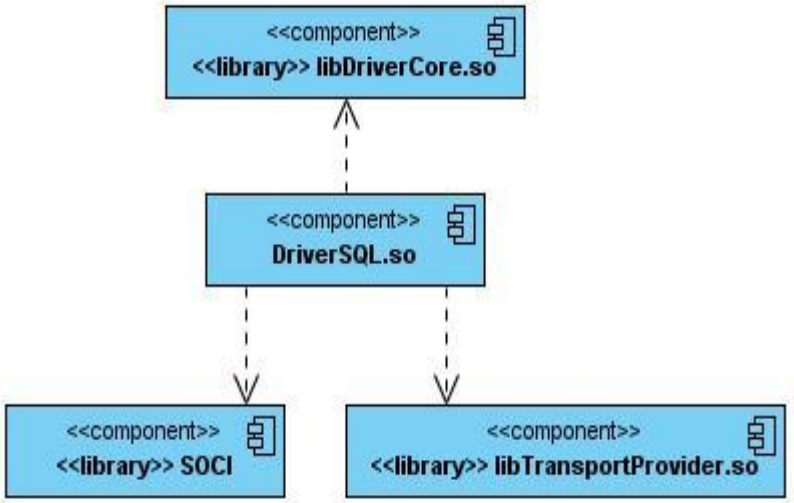

Figura 7: Diagrama de Componentes del DriverSQL.

4.5 Pruebas

Las pruebas constituyen un proceso de ejecución de programas con la intención de descubrir errores, las cuales tienen éxitos si se descubren errores que no han sido detectados hasta ese entonces.

4.5.1 Tipos de Pruebas

Para saber si un sistema funcionará correctamente o no, se llevan a cabo Pruebas de Unidad y dentro de estas las Pruebas de Caja Blanca y Caja Negra. En las Pruebas de Caja Blanca se observa siempre el código, las mismas se realizan para probarlo todo, verifican la implementación interna de la unidad y la aplicación es probada desde dentro. Por su parte, 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. Estas no tienen conocimiento del funcionamiento interno y la estructura del sistema, sino las entradas específicas con sus posibles salidas.

4.5.2 Pruebas realizadas al Manejador

Las pruebas realizadas al manejador fueron creadas con el objetivo de identificar problemas por parte de la solución, específicamente tanto en los requisitos funcionales establecidos, como por parte de la codificación. Las mismas se enfocaron en pruebas funcionales de Caja Negra para comprobar el buen funcionamiento del manejador.

- ✓ **Caso 1: Lectura de Variables.**

En las pruebas hechas al manejador se tuvo en cuenta que los manejadores en el sistema SCADA “Guardián del ALBA” están en ejecución períodos de tiempo prolongados, por lo que es necesario además de medir la capacidad de recolectar las variables especificadas previamente en la configuración, también se mida el consumo de recursos que requiere esta acción, como es el caso de si el uso de la memoria RAM se mantiene estable durante toda la ejecución de las tareas, que tiempo es capaz de mantenerse el manejador sin interrumpir su ejecución a pesar de problemas existentes como la pérdida de conexión, entre otros problemas existentes de manera negativa los cuales ocurren en los ambientes de despliegues existentes. Los resultados que se esperaban y los reales coincidieron. La lectura de variables resultó ser satisfactoria, los valores siempre fueron recuperados y en caso fueron reportados los errores correspondientes y a su vez fueron devueltos los valores de calidad de las variables que se esperaban en este caso.

Descripción general:

Las pruebas realizadas al manejador son las siguientes:

1. Leer tipos de datos STRING.

2. Leer tipos de datos ENTERO.

3. Leer tipos de datos REAL.

CP1: Leer tipos de datos String

Descripción:

Este caso de prueba obtiene el valor de una variable String, donde la calidad de la lectura deber ser 192 lo cual significa que se realizó una lectura correcta. Esta prueba se realizó con la herramienta Recolector Gráfico y la base de datos.

Pasos a seguir:

1. En la ventana “Capturas” del Recolector Gráfico se debe seleccionar la opción Crear Variable.
2. En la opción Nombre establecer un nombre a la variable creada previamente, en la opción Período establecer el valor 1000, en la opción Dirección de Lectura escribir una dirección de consulta válida en lenguaje SQL, en la opción Tipo de Dato seleccionar el tipo STRING y seleccionar aceptar.
3. En el menú Operaciones del Recolector Gráfico seleccionar la opción Comenzar Recolección.

Precondiciones para la Ejecución:

El manejador y el dispositivo previamente deben estar creados y configurados correctamente.

Iteraciones:

Valor a Leer	Resultado Esperado	Resultado de la Prueba	Observaciones
Se pide leer el valor del campo nombre ubicado en la muestra de la base de datos.	Se espera el valor del campo nombre de la muestra.	Se visualiza un valor correcto por lo que la calidad de lectura tiene un valor de 192 y se puede decir que es correcta.	Verificar que la información leída sea la misma información enviada por la base de datos.

Tabla 6: Prueba de Lectura de Tipos de Datos STRING.

CP2: Leer tipos de datos Entero

Descripción:

Este caso de prueba obtiene el valor de una variable entera, donde la calidad de la lectura deber ser 192 lo cual significa que se realizó una lectura correcta. Esta prueba se realizó con la herramienta Recolector Gráfico y la base de datos.

Pasos a seguir:

1. En la ventana “Capturas” del Recolector Gráfico se debe seleccionar la opción Crear Variable.
2. En la opción Nombre establecer un nombre a la variable creada previamente, en la opción Período establecer el valor 1000, en la opción Dirección de Lectura escribir una dirección de consulta válida en lenguaje SQL, en la opción Tipo de Dato seleccionar el tipo INT y seleccionar aceptar.
3. En el menú Operaciones del Recolector Gráfico seleccionar la opción Comenzar Recolección.

Precondiciones para la Ejecución:

El manejador y dispositivo previamente deben estar creados y configurados correctamente.

Iteraciones:

Valor a Leer	Resultado Esperado	Resultado de la Prueba	Observaciones
Se pide leer el valor del campo edad ubicada en la muestra de la base de datos.	Se espera el valor del campo edad de la muestra.	Se visualiza un valor correcto por lo que la calidad de lectura tiene un valor de 192 y se puede decir que es correcta.	Verificar que la información leída sea la misma información enviada por la base de datos.

Tabla 7: Prueba de Lectura de Tipos de Datos ENTERO.

CP3: Leer tipos de datos Real

Descripción:

Este caso de prueba obtiene el valor de una variable real, donde la calidad de la lectura deber ser 192 lo cual significa que se realizó una lectura correcta. Esta prueba se realizó con la herramienta Recolector Gráfico y la base de datos.

Pasos a seguir:

1. En la ventana “Capturas” del Recolector Gráfico se debe seleccionar la opción Crear Variable.
2. En la opción Nombre establecer un nombre a la variable creada previamente, en la opción Período establecer el valor 1000, en la opción Dirección de Lectura escribir una dirección de consulta válida en lenguaje SQL, en la opción Tipo de Dato seleccionar el tipo REAL y seleccionar aceptar.
3. En el menú Operaciones del Recolector Gráfico seleccionar la opción Comenzar Recolección.

Precondiciones para la Ejecución:

El manejador y dispositivo previamente deben estar creados y configurados correctamente.

Iteraciones:

Valor a Leer	Resultado Esperado	Resultado de la Prueba	Observaciones
Se pide leer el valor del campo nota ubicado en la muestra de la base de datos.	Se espera el valor del campo nota de la muestra.	Se visualiza un valor correcto por lo que la calidad de lectura tiene un valor de 192 y se puede decir que es correcta.	Verificar que la información leída sea la misma información enviada por la base de datos.

Tabla 8: Prueba de Lectura de Tipos de Datos REAL.

- ✓ **Caso 2: Escritura de Variables.**

Las operaciones de escritura permiten modificar valores en la base de datos. Por lo general este tipo de función solo es usada para restablecer los valores de un dispositivo, en este caso en la base de datos.

CONCLUSIONES

Como producto final se obtuvo un manejador totalmente funcional y capaz de comunicarse con Sistemas Gestores de Bases de Datos. El mismo cumple con los requisitos definidos y especificados para los manejadores del SCADA “Guardián del ALBA”. Se utilizó una arquitectura en capas conformada por la capa de Driver, capa de Protocolo, y la capa de Transporte. Se sentaron las bases para el desarrollo de nuevos manejadores.

RECOMENDACIONES

- ✓ Incluir en la Interfaz Genérica de los Manejadores la posibilidad de definir propiedades de tipo contraseña.
- ✓ Desarrollar las extensiones, para la biblioteca de acceso a datos utilizada **(SOCI)**, que permita la comunicación con Sistemas Gestores de Bases de Datos como son: Oracle y MySQL.

Bibliografía

1. **2007.** *http-peru.com.* [En línea] 2007. [Citado el: 11 de Febrero de 2010.] http://www.http-peru.com/postgresql.php.

2. **Alvarez, Rubén. 2002.** desarrolloweb.com. [En línea] 19 de Julio de 2002. [Citado el: 12 de Febrero de 2010.] http://www.desarrolloweb.com/articulos/840.php.

3. **Aragon Caceres, Jose Antonio. Llanez Jimenez, Beatriz. 2009.** *Servicio de Integración con Terceros para el Acceso a Variables del sistema SCADA "Guardián del ALBA".* Cuba : s.n., 2009.

4. **Bauta Camejo, Rene R. 2009.** *Desarrollo de una herramienta genaradora de ficheros de mapeo, para la persistencia de esquemas de objetos relacionales basada en Doctrine.* 2009.

5. cavsi.com. [En línea] [Citado el: 16 de Febrero de 2010.] http://www.cavsi.com/preguntasrespuestas/que-es-un-sistema-gestor-de-bases-de-datos-o-sgbd/.

6. **Cedeño Pozo, Antonio. 2009.** *Módulos de adquisición y análisis para la interacción con dispositivos de campo en un SCADA.* 2009.

7. **Contreras Rodríguez., Lisseidy. 2009.** *Estudio de la factibilidad del Modelo Objeto Relacional en los sistemas de gestión de información.* 2009.

8. **Dominguez Rodriguez, Alexei. 2009.** *Diseño e implementación de la Base de Datos del Sistema de planificación del entrenamiento deportivo de Judo Femenino.* 2009.

9. **2008.** eaprende.com. [En línea] 2008. [Citado el: 13 de Febrero de 2010.] http://www.eaprende.com/gestor-de-basededatos-mysql-postresql-sqlite.html.

10. **Espinosa Añel, Rafael. 2009.** *Implementación de un plugin para Alfresco que permita la firma digital de documentos.* 2009.

11. **2007.** freedownloadmanager.org. [En línea] 5 de Marzo de 2007. [Citado el: 13 de Febrero de 2010.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

12. **2007.** freedownloadmanager.org. [En línea] 5 de Marzo de 2007. [Citado el: 16 de Febrero de 2010.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p.

13. **García Hernández, Luis Enrique. 2009.** *Framework para el desarrollo de manejadores de dispositivos, para el proyecto SCADA Guardián del ALBA.* 2009.

14. **Gonzalez Blanco. Ruben, Perez Tobalina. Sergio.** essi.upc.edu. [En línea] [Citado el: 12 de Febrero de 2010.]

http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=3&ved=0CA4QFjAC&url=http%3A%2F%2Fwww.essi.upc.edu%2F~es-e%2Fweb%2Fdocuments%2Flab%2F0304Q2%2Flessons%2Flese-2%2FLESE-2%2520-%2520Introduccion%2520a%2520Rational%2520Rose.ppt&rct=j&q=rational+rose+cara.

15. **Guerra Garayta, Ariel. 2009.** *WebUInt, Interfaz Hombre-Máquina del SCADA en la Web.* 2009.

16. **Hernandez Perdomo, Mariluz. Gonzalez Fernandez, Enrique Jose. 2009.** *Guía para la optimización de servidores de bases de datos de PostgreSQL.* 2009.

17. **2004.** informatica.uv.es. [En línea] 2004. [Citado el: 11 de Febrero de 2010.] http://informatica.uv.es/it3guia/LP/laboratorio/P1/TutorialEclipse.pdf.

18. **2007.** isca. [En línea] Actimedia Digital, 2007. [Citado el: 11 de Febrero de 2010.] http://www.isca.com.ve/det_productos.php?Id=10.

19. **Leyet Fernandez, Osmar. Rodriguez Lorenzo, Iosmel. 2008.** *Desarrollo de una herramienta genaradora de ficheros de mapeo para la persistencia de esquemas de objetos relacionales basada en NHibernate.* 2008.

20. **2006.** mailxmail.com. [En línea] 24 de Febrero de 2006. [Citado el: 10 de Febrero de 2010.] http://www.mailxmail.com/curso-procesamiento-datos-oracle/sistema-manejador-base-datos.

21. **2004.** mastermagazine.info. [En línea] 2004. [Citado el: 13 de Febrero de 2010.] http://www.mastermagazine.info/termino/7006.php.

22. **Mató Rodriguez, Raúl. 2009.** *Diseño de Bases de Datos para la Empresa de Gas Manufacturado.* 2009.

23. netpecos.org. [En línea] [Citado el: 11 de Febrero de 2010.] http://www.netpecos.org/docs/mysql_postgres/x57.html.

24. **Páez Castillo, Douglas. 2009.** *Desarrollo de la arquitectura de la Plataforma de Televisión Informativa PRIMICIA.* 2009.

25. **Pimienta Fernández, Yunier Alexander. 2009.** *Propuesta de Arquitectura de Software para la Empresa de Perforación y Extracción de Petróleo de Occidente.* 2009.

26. **Ruz Pérez, Yelaine. 2009.** *Propuesta de arquitectura para agilizar el desarrollo de sistemas de gestión web sobre portales empresariales libres.* 2009.

27. sharewareconnection.com. [En línea] [Citado el: 16 de Febrero de 2010.] http://www.sharewareconnection.com/visual-paradigm-for-uml-professional-edition.htm.

28. sourceforge.net. [En línea] [Citado el: 19 de Febrero de 2010.] http://soci.sourceforge.net/.

29. **2010.** sourceforge.net. [En línea] 2010. [Citado el: 19 de Febrero de 2010.] http://sourceforge.net/apps/trac/litesql/.

30. **2010.** sqlapi.com. [En línea] 10 de Febrero de 2010. [Citado el: 19 de Febrero de 2010.] http://www.sqlapi.com/.

31. **Tabares Cuevas, David. 2009.** *Analisis y Diseño del Sistema de Manejo Integral de Perforación de Pozos.* 2009.

32. **Trujillo, Rafael. 2006.** *Especificación de Requerimientos de SoftwareDel Modulo de Drivers.* 2006.

ANEXOS

Anexo 1: Proceso de Desarrollo de Software.

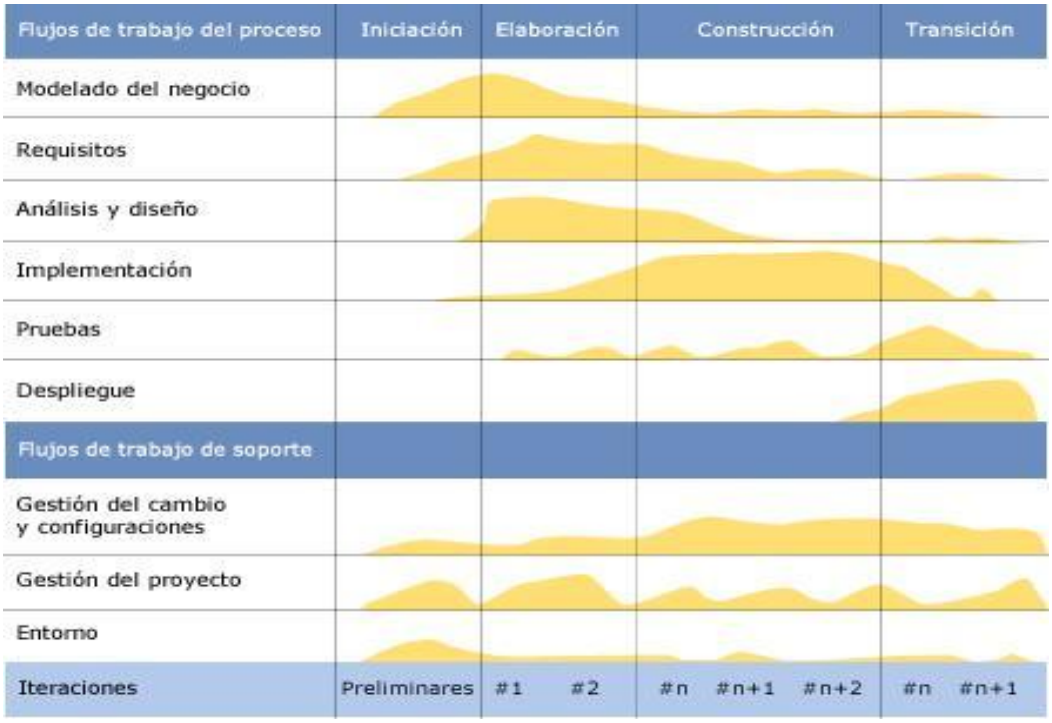

Figura 8: Proceso de Desarrollo de Software.

Anexo 2: Clase del Diseño SqlDevice.

Nombre: SqlDevice	
Tipo de clase:	
Atributo	Tipo
provider	ITransportProvider*
connString	ConnectionString*
startBlockAddress	long long
usedRegistries	long long
endPoint	SqlEndPoint*

snifferHandler	SnifferHandler
startBlockVarType	DeviceVarType
Para cada responsabilidad:	
Nombre:	SqlDevice()
Descripción:	Constructor por Defecto.
Nombre:	getDataBaseName()
Descripción:	Función que devuelve una cadena con el nombre de la base de datos.
Nombre:	setDataBaseName()
Descripción:	Función que establece el valor de una cadena con el nombre de la base de datos.
Nombre:	resetDiagnosticParameters()
Descripción:	Restablece los parámetros de diagnóstico del manejador.
Nombre:	createBlock()
Descripción:	Función que crea un Bloque de variables.
Nombre:	startBlock()
Descripción:	Función que establece una dirección como dirección de comienzo de un bloque.
Nombre:	createIProtocolAddress()
Descripción:	Función que crea una instancia de IProtocolAddress a partir de la representación textual de la dirección dada por el parámetro Address.
Nombre:	fitInBlock()
Descripción:	Función que determina si el rango de direcciones puede alojarse en un bloque del protocolo.
Nombre:	fitInBlockSingle()
Descripción:	Función que determina si una variable puede alojarse en un bloque.
Nombre:	compare()
Descripción:	Función que compara dos direcciones de protocolo de acuerdo a las reglas definidas por este.

Nombre:	setAddress()
Descripción:	Función que establece el valor de la dirección.
Nombre:	getAddress()
Descripción:	Función que devuelve el valor de la dirección.
Nombre:	getCOMMSTATE()
Descripción:	Función que devuelve el estado de la comunicación.
Nombre:	getEndPoint()
Descripción:	Función que devuelve la instancia del EndPoint.
Nombre:	incrementFailure()
Descripción:	Función que devuelve la cantidad de fallos.
Nombre:	incrementSucess()
Descripción:	Función que devuelve la cantidad de éxitos.
Nombre:	setDbServerAdress()
Descripción:	Función que establece el valor de dbServerAdress.
Nombre:	setDbServerPort()
Descripción:	Función que establece el puerto de conexión a la BD.
Nombre:	setDbUser()
Descripción:	Función que establece el usuario de conexión a la BD.
Nombre:	setDbPassword()
Descripción:	Función que establece la contraseña de conexión a la BD.
Nombre:	setConnectTimeout()
Descripción:	Función que establece el tiempo de espera.
Nombre:	setDbBackend()
Descripción:	Función que establece el backend de la BD.

Nombre:	getTransportProvider()
Descripción:	Función que devuelve una instancia del TransportProvider.

Tabla 9: Clase del Diseño SqlDevice.

Anexo 3: Clase del Diseño SqlDeviceMetaClass.

Nombre: SqlDeviceMetaClass	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	SqlDeviceMetaClass()
Descripción:	Constructor por Defecto.
Nombre:	registerProperties()
Descripción:	Función que registra las propiedades de configuración y parámetros de diagnóstico que almacenan las metaclases.

Tabla 10: Clase del Diseño SqlDeviceMetaClass.

Anexo 4: Clase del Diseño SqlDriverMetaClass.

Nombre: SqlDriverMetaClass	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	SqlDriverMetaClass()
Descripción:	Constructor por Defecto.
Nombre:	registerProperties()
Descripción:	Función que registra las propiedades de configuración y parámetros de diagnóstico que almacenan las metaclases.
Nombre:	internalCreateInstance()
Descripción:	Función que responde por la creación de una instancia de la clase Driver.

Tabla 11: Clase del Diseño SqlDriverMetaClass.

Anexo 5: Crear Manejador en el Recolector Gráfico.

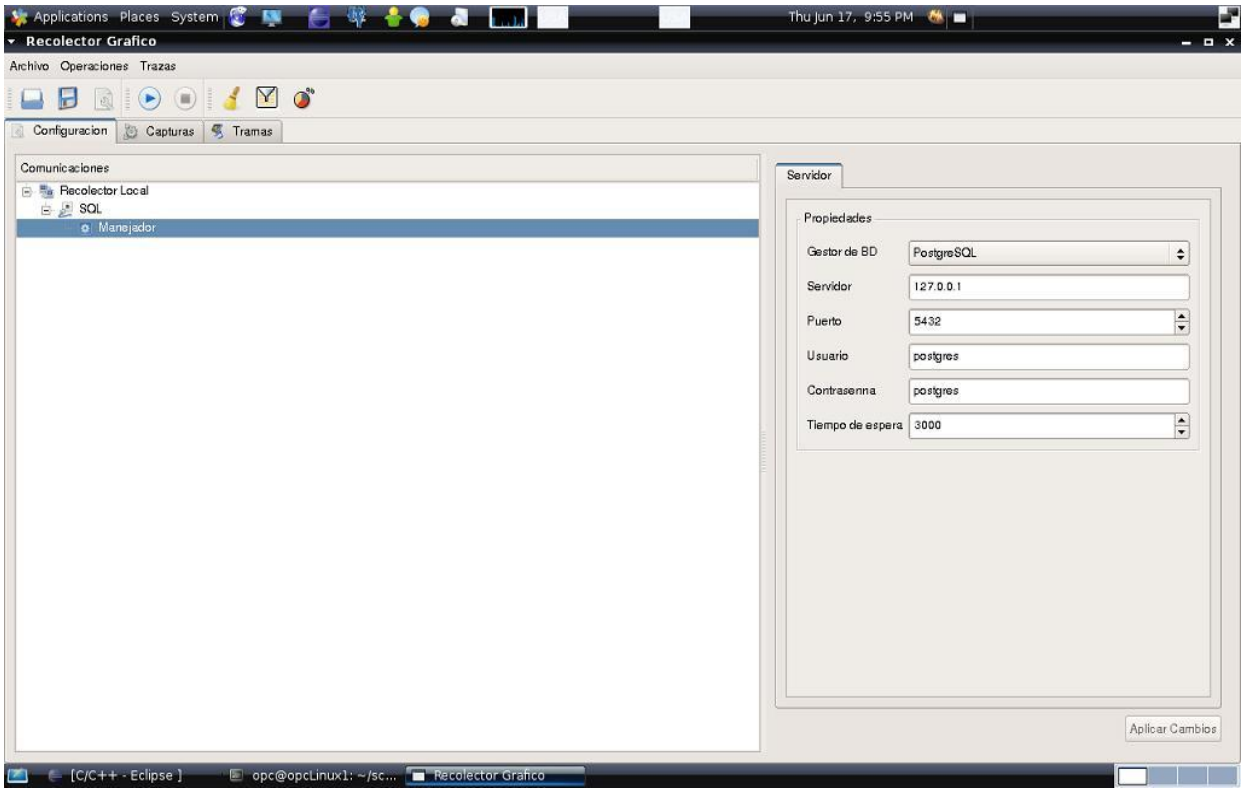

Figura 9: Crear Manejador en el Recolector Gráfico.

Anexo 6: Crear Dispositivo en el Recolector Gráfico.

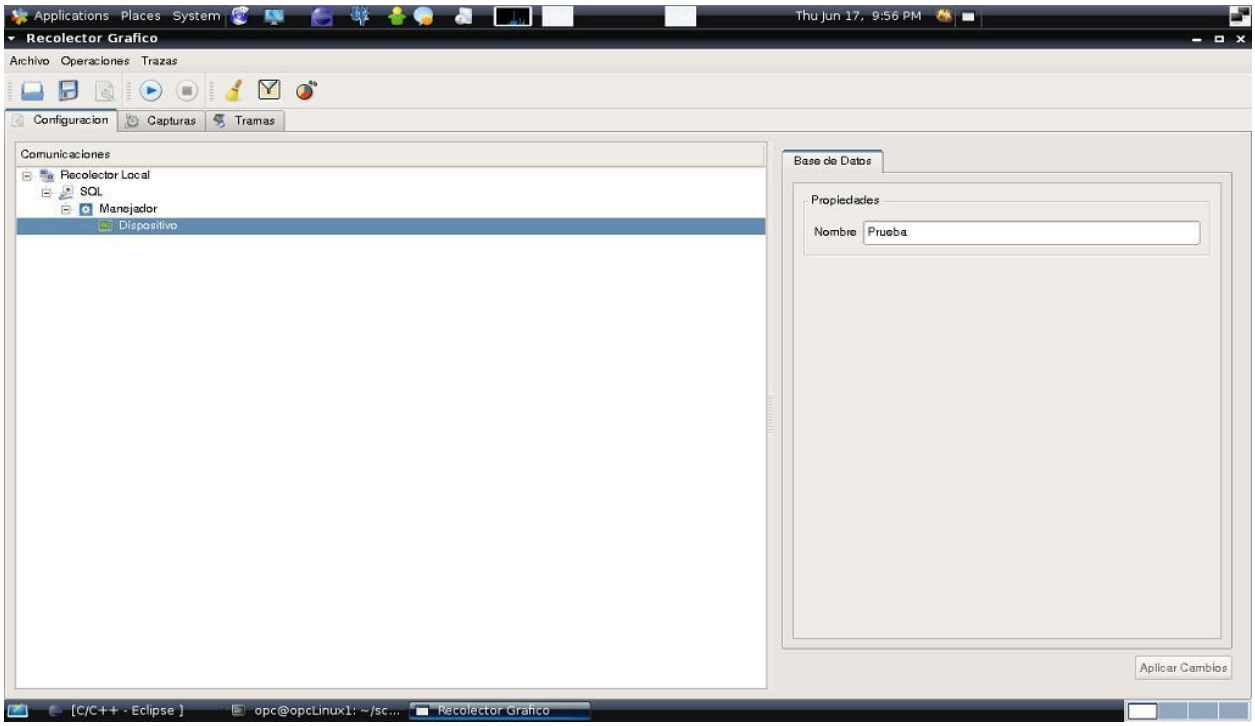

Figura 10: Crear Dispositivo en el Recolector Gráfico.

GLOSARIO DE TÉRMINOS

A:

ACID *(**A**tomicity, **C**onsistency, **I**solation and **D**urability):* conjunto de reglas y restricciones que se establecen como forma de unidad de trabajo atómica. Este estándar incluye características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción.

B:

Backend: Son sistemas o elementos de un sistema de mantenimiento o administración (en este caso, de la base de datos) y, por tanto, no accesibles a las interfaces públicas (front-ends).

P:

Plug-in: (también conocido como *addin, add-in, addon o a dd-on*) es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica, Esta aplicación adicional es ejecutada por la aplicación principal.

S:

Select: Sentencia del lenguaje SQL que permite consultar registros o tuplas.

T:

Tuplas: Son los registros que se almacenan en una tabla, identificado de forma única por su clave primaria y con un conjunto de atributos que lo diferencian de los demás.

U:

Update: Sentencia del lenguaje SQL que permite modificar registros o tuplas.