Universidad de las Ciencias Informáticas
Facultad 5

MEJORAS AL ESQUEMA DE RÉPLICA DEL SERVIDOR REDUNDANTE DEL MÓDULO DE ADQUISICIÓN DEL SCADA GALBA

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

Autor(es): Yairilee Cruz Castillo

Tutor(es): Ing. Yunior Peralta González

Ing. Juan Carlos González Tamayo

Co-Tutor: Ing. Rafael Alejandro Pérez Ordoñez

La Habana, 30 de junio de 2015
"Año 57 de la Revolución"

"El genio comienza las grandes obras, pero sólo el trabajo las acaba"

Joseph Joubert

DECLARACIÓN DE AUTORÍA

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

Para que así conste, firmamos la presente a los _____ días del mes de _____ Junio del año 2015.

_____

Firma del Autor
Yairilee Cruz Castillo

_____ _____

Firma del Tutor
Ing. Yunior Peralta González

Firma del Tutor
Ing. Juan Carlos González Tamayo

_____

Firma del Co-Tutor
Ing. Rafael Alejandro Pérez Ordoñez

DATOS DE CONTACTO

Nombre y apellidos del tutor(a): Ing. Yunior Peralta González
Institución: Universidad de las Ciencias Informáticas.
Título: Ingeniero en Ciencias Informáticas.
Correo: yperalta@uci.cu

Nombre y apellidos del tutor(a): Ing. Juan Carlos González Tamayo
Institución: Universidad de las Ciencias Informáticas.
Título: Ingeniero en Ciencias Informáticas.
Correo: jctamayo@uci.cu

Nombre y apellidos del tutor(a): Ing. Rafael Alejandro Pérez Ordoñez
Institución: Universidad de las Ciencias Informáticas.
Título: Ingeniero en Ciencias Informáticas.
Correo: rafaelalejandro@uci.cu

AGRADECIMIENTOS

Agradecer a mis padres y hermanos por su sacrificio y dedicación, por estar siempre a mi lado apoyándome en las buenas y en las malas.

A mis abuelos, tíos, primos y demás familiares por estar siempre presente brindándome su apoyo incondicional.

A mi novio que ha soportado todos mis cambios de humor que han sido muchos en todos estos años, por la comprensión, amor y respesto demostrado. No creo que hubiese alcanzado resultado alguno sin tu ayuda , te amo.

A mis amistades, por estar ahí y haber contribuido a ser la persona que soy hoy, especialmente a Mailin por ser mi amiga, y aunque no esté lejos se que puedo contar con ella.

A mis suegros Coralia y Félix por estar siempre a mi lado apoyándome en todo momento, dándome su amor y confianza.

A Rafael, mi co-tutor, por ayudarme en todo momento, por en estar ahí cuando más lo necesitaba, por pasar malas noches ayudándome en el desarrollo de la tesis, que además de tutor ha sido un amigo en las buenas y las malas. Esta tesis no se habría concretado de no ser por él….es por eso que si hoy soy ingeniera se lo debo a él.

A Yunior, mi tutor, por la paciencia, que sé que conmigo hay que tener mucha.

A Juan Carlos, mi tutor, por todos los conocimientos transmitidos aunque no nos hemos conocido personalmente he sabido apreciarte.

A mis compañeros, por los mejores años de mi vida.

DEDICATORIA

Quiero dedicarle el presente trabajo de diploma especialmente a mis padres y hermano los cuales han sabido guiarme, aconsejarme hasta el final de esta carrera sin ellos no fuera posible, a mi novio por estar a mi lado todos estos años ayudándome en los buenos y malos momentos, a todas esas personas que me apoyaron a conseguir mi sueño, graduarme.

RESUMEN

El módulo para la Adquisición de datos del sistema SCADA Guardián del Alba en su versión Miranda R2 presenta entre sus principales funcionalidades la de recolectar los valores de los puntos de los dispositivos de campo. Este módulo puede ser desplegado en un servidor de respaldo con requisitos de alta disponibilidad para garantizar el funcionamiento del sistema ante una falla en el servidor principal. Para esto el servidor de respaldo debe mantenerse actualizado con los cambios que ocurran en el servidor principal. Sin embargo en la versión actual, debido al tiempo que demora la sincronización de los datos entre estos dos servidores, en ocasiones el servidor de respaldo se encuentra desactualizado por más de quince segundos. Para darle solución a este problema, el presente trabajo modificó el esquema de la sincronización de datos del módulo de Adquisición logrando que el tiempo máximo de desactualización del servidor de respaldo no exceda de un segundo.

Palabras clave: adquisición de datos, alta disponibilidad, réplica, SCADA.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

INTRODUCCIÓN

En el amplio campo de las Tecnologías de la Información y las Comunicaciones (TIC), tiene lugar el uso de sistemas o elementos de cómputo en el control automatizado de procesos industriales, sustituyendo así a operadores humanos. La automatización de estos procesos nace de la necesidad de adquirir, almacenar y visualizar la información de los dispositivos; de manera que surgen sistemas que permiten supervisar y controlar variables de proceso a distancia denominados SCADA (acrónimo de Supervisión, Control y Adquisición de Datos). De forma general, SCADA es un término que se utiliza para describir la supervisión, control y gestión de soluciones en una amplia gama de industrias, por ejemplo en sistema de gestión de agua, energía eléctrica, señales de tráfico, sistemas de control ambiental, y sistemas de fabricación. (1)

Desde el punto de vista del *software*, un SCADA, se puede definir como una aplicación diseñada para funcionar sobre un conjunto de ordenadores que proporcionan comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, etc.) y permiten controlar la producción. Además, provee a diversos usuarios la información que se genera en el proceso productivo: control de la calidad, supervisión, mantenimiento. En este tipo de sistemas usualmente existe un ordenador que efectúa tareas de supervisión, gestión de alarmas, tratamiento de datos y control de procesos. (2)

Los SCADA se pueden dividir en varios niveles, cada nivel se encarga de una parte de las funcionalidades. Normalmente, en los niveles más bajos del SCADA, se incluye el *hardware* de instrumentación y actuación que es el núcleo de las señales con las que opera el proceso. Este *hardware* puede ser conectado a un sistema recolector de información que sirve de interfaz a las capas de procesamiento de alto nivel para tomar decisiones ya sean de forma automática o por medio de operadores. La información del proceso es mostrada a los operadores por medio de interfaces de usuario (HMI[1]), a través de las cuales se puede ejercer control sobre la planta mediante la ejecución de comandos o la modificación de las configuraciones del sistema. (2)

La Empresa Socialista de Capital Mixto Guardián del Alba (ESCMGA) formada por Albet S.A. y Petróleos de Venezuela S.A. (PDVSA) desarrolla y despliega el sistema SCADA “Guardián del Alba” (Galba). Albet tiene un contrato con el Centro de Informática Industrial (CEDIN), perteneciente a la Facultad 5 de la Universidad de las Ciencias Informáticas (UCI), con el objetivo de desarrollar, mantener y desplegar el

[1] HMI: Interfaz Hombre-Máquina.

sistema SCADA Galba. Este sistema presenta una arquitectura distribuida, la cual consta de una serie de módulos interconectados a través de una capa de comunicaciones conocida como *middleware*. Ejemplo de estos módulos son: Configuración, Históricos, HMI y Adquisición.

El módulo de Adquisición es el módulo más importante del sistema, en él se integran la recolección y el procesamiento de datos. Como recolector es el encargado de la gestión de los *drivers* y de la planificación de la encuesta a los dispositivos. Durante el procesamiento analiza las alarmas configuradas, publica los resultados a los demás módulos y salva la información adquirida para su posterior uso por otras aplicaciones. Interactúa fundamentalmente con el *middleware* para la comunicación con el resto de los módulos. (3)

Este módulo puede ser desplegado en un segundo servidor para garantizar que al ocurrir un fallo en el servidor principal, los servicios que este ofrecía sean atendidos por el servidor de respaldo, garantizando así una alta disponibilidad del sistema. En este caso de intercambio de servidores es importante que el estado de la información brindada se encuentre actualizada, por ejemplo: en caso de que el servidor principal haya modificado parte de su información y cinco segundos después falle, entonces el servidor de respaldo al tomar el control debe presentar esta modificación, en caso contrario, se dice que el servidor de respaldo se encuentra desactualizado con un tiempo de al menos cinco segundos.

El módulo de Adquisición se encuentra dividido por varios procesos que se comunican entre sí utilizando memorias compartidas. Para mantener actualizado al servidor de respaldo, el servidor principal le envía la información almacenada en estas memorias de forma periódica, siendo este período igual al tiempo que demora el envío, aumentado en un tiempo dependiente de la configuración del sistema cuyo valor mínimo es de diez segundos. Debido a esto el tiempo que puede estar desactualizado el servidor de respaldo no excede a este período, el cual puede utilizarse para estimar el tiempo máximo de desactualización del servidor de respaldo.

El período de la réplica varía de forma proporcional a la magnitud del proceso que se esté supervisando, por ejemplo: para procesos con más de veinte mil puntos, este período puede alcanzar los quince segundos, provocando que cambios ocasionados en el servidor principal no se reflejen en el de respaldo hasta después de transcurrir este tiempo, lo cual, en dependencia de la criticidad del proceso que se esté supervisando,

puede ser considerado un error por no cumplir con requisitos referentes a la actualización del sistema en tiempo real.

Además, el tiempo de desactualización puede provocar inconsistencias en varios módulos del sistema SCADA, por ejemplo: en el servidor de base de datos históricos pueden arribar muestras repetidas; y en las consolas de HMI pueden aparecer representaciones incorrectas en las gráficas de tendencias de tiempo real, y visualizaciones de alarmas que no se encuentran presentes en el servidor redundante de Adquisición. Otro de los problemas de inconsistencia presentados se encuentra en el módulo de Adquisición del servidor de respaldo, debido a que el mismo obtiene su información de las memorias compartidas del servidor principal mientras este las modifica. Estas acciones ejecutadas en paralelo se realizan sin tener en cuenta ninguna de las técnicas para la sincronización entre procesos, por lo que la información replicada puede, en algunos casos, presentar problemas de inconsistencia que pudieran ocasionar un mal funcionamiento en el servidor de respaldo.

Debido a que mientras mayor sea el tiempo de la desactualización mayor es la cantidad de inconsistencias, y al hecho de que es desfavorable para el sistema no estar actualizado, se define como **problema de la investigación**: ¿Cómo disminuir el tiempo de la desactualización en el servidor de respaldo del módulo de Adquisición de la versión Miranda R2 del sistema SCADA Galba?

Una vez planteado el problema de la investigación, se deriva como **objeto de estudio**: los sistemas para la alta disponibilidad.

Se persigue como **objetivo general** de la investigación: Modificar el esquema de réplica en el módulo de Adquisición de la versión Miranda R2 del sistema SCADA Galba para disminuir el tiempo de la desactualización en el servidor de respaldo.

Definiéndose como **campo de acción**: la réplica entre dos servidores de adquisición de datos de sistemas SCADA. De acuerdo con el problema planteado **la idea a defender** es la siguiente: si se modifica el esquema de réplica en el módulo de Adquisición de modo que al arribar la información al servidor principal, ésta se envíe para el servidor de respaldo, entonces se disminuye el tiempo de desactualización del servidor de respaldo.

Para dar solución al objetivo propuesto se definen las siguientes **Tareas de Investigación**:

1) Revisión del estado del arte de sistemas que utilicen servidores redundantes, para identificar aspectos a tener en cuenta en la implementación del nuevo esquema de réplica del módulo de Adquisición.
2) Estudio y caracterización del módulo de Adquisición para identificar aspectos relevantes de su diseño e implementación a tener en cuenta en el desarrollo de la presente investigación.
3) Definición de los requisitos funcionales y no funcionales.
4) Actualización del modelo de diseño en correspondencia con los nuevos requisitos.
5) Implementación de los requisitos definidos.
6) Diseño y ejecución de pruebas para comprobar el cumplimiento de los requisitos y del objetivo de la investigación.

Para el cumplimiento de estos objetivos se utilizan varios métodos y técnicas en la búsqueda de información, los cuales son:

Métodos teóricos:

- ✓ Método analítico-sintético: Se emplea para el estudio de los conceptos en el módulo de Adquisición del SCADA realizando un análisis de los documentos elaborados por los desarrolladores, analistas y asesores automáticos, para la extracción y síntesis de los elementos más importantes.
- ✓ Análisis histórico–lógico: Se emplea para realizar el estudio del estado del arte acerca del tema en cuestión, analizando los antecedentes y las tendencias actuales en cuanto a la evolución y desarrollo de los sistemas redundantes.
- ✓ Método modelación: Se emplea en el diseño de clases y algoritmos que intervienen en la solución del problema planteado permitiendo reproducirlos mediante diagramas, tablas y nuevos conceptos.

Métodos empíricos:

- ✓ Entrevistas: Se utilizan para la recolección de la información y el conocimiento existente en los especialistas vinculados a los temas de automatización industrial.

- ✓ Experimentos: Empleados en la elaboración de prototipos funcionales, con el objetivo de comprobar la efectividad de la implementación de las funcionalidades.

El presente documento está estructurado en tres capítulos:

Capítulo 1: “Fundamentación teórica”.
Se fundamenta el marco teórico de la investigación relacionado con el sistema SCADA Galba. Se describen los sistemas de réplica estudiados, la metodología y las herramientas de desarrollo a utilizar.

Capítulo 2: “Análisis y Diseño”.
Se presenta la propuesta de solución utilizando el modelo conceptual de la aplicación, se definen los requisitos mediante las historias de usuario y se describe el modelo de diseño.

Capítulo 3: “Implementación y Pruebas”.
Se valida la solución propuesta mediante pruebas de caja blanca, caja negra y de rendimiento. Además, se valida el diseño mediante las métricas TOC y RC.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se introducen los conceptos fundamentales relacionados con el sistema SCADA Galba. Además se describen las tecnologías, herramientas y metodología a utilizar para el desarrollo de la investigación.

1.1 Principales conceptos

1.1.1 SCADA Guardián del Alba

El SCADA Galba es un sistema distribuido en módulos que trabajan de manera conjunta posibilitando el funcionamiento del sistema como un todo. Uno de sus módulos es el de Adquisición, a continuación se describe su funcionamiento:

1.1.2 Módulo de Adquisición

El módulo de Adquisición es el responsable de recolectar, procesar y almacenar la información del proceso productivo que supervisa. Está dividido en diferentes procesos que se encuentran en constante ejecución. Uno de ellos, el recolector, utilizando los manejadores de protocolos obtiene la información de los dispositivos de campo ubicados a distancias remotas. Esta información se procesa y se almacena en memorias compartidas, permitiendo ser accedidas por los procesos del módulo de Adquisición. Luego esta información se publica hacia las consolas HMI y a la base de datos histórica (HDB). (2)

Se describen a continuación los principales recursos que se gestionan en el módulo de Adquisición:

Puntos o Variables: Son valores que se recolectan de los dispositivos de campo. Constituyen la representación de la información básica del sistema SCADA Galba. Estos puntos pueden tomar los valores de temperatura, presión, intensidad de corriente, carga, voltaje, resistencia, volumen, velocidad, entre otros. Un punto puede ser analógico o digital. El analógico puede tomar valores entre dos valores cualesquiera de su dominio, como por ejemplo la presión y la temperatura, mientras que el digital puede tomar valores dentro de un conjunto discreto, ejemplo de ello, un dispositivo encendido o apagado. (4)

Alarmas: Representan condiciones anormales del proceso bajo supervición. En ocaciones requieren la atención de un operador para darle solución antes de que se llegue a una situación crítica que detenga el proceso. (2)

Dispositivos: Se denomina dispositivo a un elemento de hardware que alberga información de proceso o estado de sí mismo, que forma parte de un sistema automatizado. Comúnmente los dispositivos son Controladores Lógicos Programables, Computadoras Industriales, Sistemas de Control Distribuidos, sensores o actuadores inteligentes con capacidad de comunicación. (2)

Comandos: Son acciones correctivas que se ejecutan sobre los procesos que se supervisan. Se envían desde las consolas HMI hacia el módulo de Adquisición. Se aplican sobre los recursos puntos, alarmas y dispositivos, y permiten que dichos recursos sean sincronizados al módulo HMI. (2)

1.1.3 Memorias compartidas

Las memorias compartidas pueden ser accedidas por múltiples procesos. Éstas permiten a dos o más procesos compartir un segmento de memoria, y por consiguiente, los datos que hay en él.

En el SCADA Galba las memorias compartidas están divididas en los siguientes grupos:
Las memorias compartidas de configuración: donde se guardan las configuraciones de los puntos, alarmas y dispositivos.

Las memorias compartidas de datos: donde se guardan las últimas 8 muestras de los puntos, alarmas, estado de las comunicaciones de los dispositivos y estado de las comunicaciones de los subcanales.

1.1.4 Réplica de datos

La replicación es la operación de trasladar los datos que se estén enviando desde un servidor hacia otros, de forma que cualquiera de ellos puede entregar los mismos resultados a sus clientes. La réplica puede ser de dos tipos: “maestro-esclavo y maestro-maestro”; el primero permite al servidor maestro realizar los procedimientos de lectura/escritura, mientras que los servidores esclavos efectúan las operaciones de lectura. En el caso de maestro-maestro permite tener servidores maestros que interactúan entre sí, enviando consultas de lectura/escritura a múltiples servidores maestros.

Para replicar los datos almacenados en el módulo de Adquisición del SCADA Galba se utiliza un servidor principal y un servidor redundante. El servidor principal recolecta y procesa la información proveniente de los dispositivos de campo, la almacena en las memorias compartidas y la publica a sus clientes.

1.1.5 Alta disponibilidad

La alta disponibilidad consiste en una serie de medidas para garantizar la disponibilidad del servicio, es decir, asegurar que el servicio funcione durante las veinticuatro horas. El término disponibilidad hace referencia a la probabilidad de que un servicio funcione adecuadamente en cualquier momento. (5)

Es el conjunto de dos o más máquinas que comparten servicios y que se monitorizan constantemente entre sí. Es un protocolo de diseño del sistema y su implementación asegura un cierto grado de continuidad operacional durante un período de medición dado. (6)

Alta Disponibilidad está conformada por los módulos:

- Sistema Monitor.
- Redundancia y Respaldo.

Sistema Monitor

El monitor tiene la labor de prestar alta disponibilidad de servicios, para ello ejecuta los servicios configurados en cada nodo y chequea el estado de cada uno. (7)

Redundancia y Respaldo

Los SCADA, deben funcionar 24 horas al día, los 7 días de la semana, por lo que es indispensable que los servicios estén siempre disponibles de manera correcta.

Para ello es necesario que exista un servidor de respaldo que permita dar funcionalidad de servicios cuando el servidor que estaba activo falle.

Este módulo es configurable y posee dos servidores conectados entre sí, lo cual permite que en caso de que el servidor principal falle, los servicios migren al servidor de respaldo para así garantizar que este último ofrezca los servicios mientras el otro se encarga de solucionar la falla. (7)

1.1.6 Configuraciones de alta disponibilidad

Las configuraciones más habituales en entornos de alta disponibilidad son las configuraciones activo/activo y activo/pasivo. (6)

- **Configuración Activo/Activo**

En una configuración activo/activo, todos los servidores pueden ejecutar los mismos recursos simultáneamente. Los servidores poseen los mismos recursos y pueden acceder a estos independientemente de los otros servidores. Si un servidor falla y deja de estar disponible, sus recursos siguen estando accesibles a través de los otros servidores.

La principal ventaja de esta configuración es que los servidores son más eficientes ya que pueden trabajar todos a la vez. Pero cuando uno de los servidores deja de estar disponible su carga de trabajo pasa a los servidores restantes, esto produce una degradación en el servicio ofrecido.

En la Figura 1 se muestra como ambos servidores están activos, proporcionando un mismo servicio a los diferentes usuarios. Los clientes acceden al servicio o recursos de forma transparente y no tienen conocimiento de la existencia de varios servidores. (6)

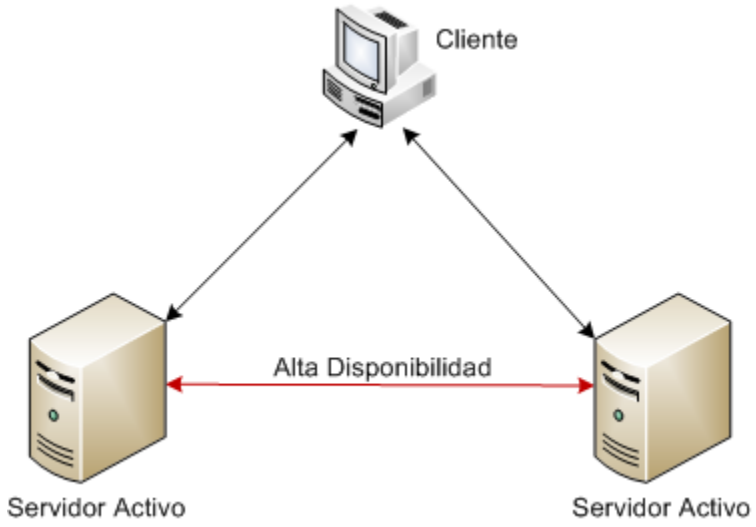

Figura 1: Ejemplo de Alta Disponibilidad con configuración activo/activo

- **Configuración Activo/Pasivo**

La configuración activo/pasivo consiste en un servidor principal que posee los recursos y otros servidores secundarios que son capaces de acceder a estos recursos, pero no toman el control de los mismos hasta que el propietario de los recursos ya no está disponible.

Esta configuración tiene la ventaja de que no hay degradación de servicios, los cuales solo se reinician cuando el servidor principal deja de estar disponible. La principal desventaja de este sistema con respecto al anterior, es que todos los servidores secundarios están ociosos a la espera del fallo del servidor principal, haciendo que esta solución sea menos eficiente.

En caso de que un servidor ya no esté disponible, dañado o fallido se incorpora en estado de *failover*, es decir, en estado de migración de recursos. (6)

En el SCADA Galba la configuración de alta disponibilidad que se efectúa es activo/pasivo donde existe un servidor principal y un servidor de respaldo (ver Figura 2).

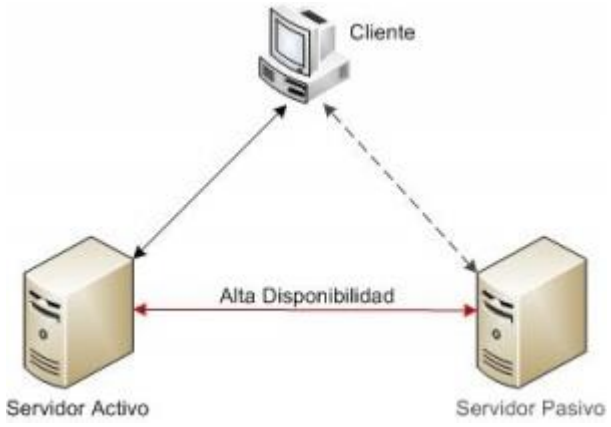

Figura 2: Ejemplo de Alta Disponibilidad con configuración activo/pasivo

1.2 Sistemas que utilizan réplica

MySQL Replication. Consiste en una replicación asíncrona unidireccional: un servidor actúa como maestro y uno o más actúan como esclavos. La replicación en MySQL se basa en un servidor maestro que registra los cambios en las bases de datos (actualizaciones, eliminaciones, inserciones) en logs binarios. Estos logs sirven como registros de actualizaciones para enviar a los servidores esclavos. Cuando un esclavo se conecta al maestro, le informa la posición hasta donde ha leído los logs en la última actualización

satisfactoria. El esclavo recibe cualquier actualización que ha tenido lugar desde entonces, se bloquea y espera a que el maestro le envíe las nuevas actualizaciones. (8)

PostgreSQL. Utiliza un sistema de replicación “maestro-esclavo”, donde el servidor maestro primero realiza un *backup* base y luego se mantiene transfiriendo registros WAL (*Write Ahead Log*) al servidor esclavo. Estos registros almacenan la información sobre las transacciones y cambios realizados en las bases de datos, y se emplean para reparar automáticamente posibles inconsistencias que puedan ocasionarse en las mismas en caso de un fallo del servidor maestro, garantizando así la integridad del servidor. (9)

Slony-I. Es un sistema de replicación “maestro a múltiples esclavos” para PostgreSQL, el cual soporta el modo de réplica en “cascada” (un nodo puede proveer a otro nodo, el cual puede proveer a otro) y *failover*. Incluye las principales características para replicar bases de datos de gran información a un número razonable de esclavos. Es por tanto un sistema diseñado para centros de datos y sitios de *backup* donde el funcionamiento normal requiere que todos los nodos estén disponibles en un momento determinado. Las principales ventajas que tiene Slony-I sobre la réplica de PostgreSQL son: interacción entre servidores con distintas versiones de PostgreSQL y replicación de una parte del contenido del servidor. (10)

1.3 Metodología de Desarrollo: AUP-UCI

Las metodologías del desarrollo del software son un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar una aplicación. Para la realización de la aplicación se utilizó: (11)

AUP-UCI

El Proceso Unificado Ágil (AUP), es una versión simplificada de Proceso Racional Unificado (RUP). Describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. (12)

Fases AUP-UCI

- **Inicio:** Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo, costo y decidir si se ejecuta o no el proyecto.

- **Ejecución:** En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del software.
- **Cierre**: En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto. (12)

Disciplinas AUP-UCI

- Modelado de negocio (opcional).
- Requisitos.
- Análisis y diseño.
- Implementación.
- Pruebas internas.
- Pruebas de Liberación.
- Pruebas de Aceptación.
- Despliegue (opcional).

1.4 Lenguaje de Modelado: UML

El Lenguaje Unificado de Construcción de Modelos (UML por sus siglas en inglés), es un lenguaje de modelado utilizado para especificar, visualizar y construir los artefactos de un sistema de software. Este modelado visual es independiente del lenguaje de implementación, de tal forma que los diseños realizados, usando UML, se puedan implementar en cualquier lenguaje que soporte sus posibilidades. Es una de las herramientas de modelado más utilizadas para la definición de la arquitectura de una aplicación, que utiliza una gran variedad de diagramas (clases, secuencias, actividades, componentes, etc). (13)

1.5 Lenguaje de Programación: C++

Es un lenguaje que se puede utilizar tanto para escribir software de bajo nivel, como *drivers* y componentes de sistemas operativos. Los compiladores de C++ generan código nativo con un alto grado de optimización en memoria y velocidad, lo que lo convierte en uno de los lenguajes más eficientes. (14)

En la actualidad, C++ constituye un lenguaje versátil, potente y general. Una particularidad de C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales.

Principales características:

- ✓ Es un lenguaje orientado a objetos.
- ✓ Es potente en lo que se refiere a creación de sistemas complejos debido a su robustez.
- ✓ Permite la separación de un programa en módulos que admiten compilación independiente.

1.6 La herramienta CASE: Visual Paradigm

Visual Paradigm es una herramienta CASE[2] multiplataforma que propicia un conjunto de ayudas para el desarrollo de programas informáticos desde la planificación, análisis y diseño, hasta la construcción, prueba y despliegue. Permite realizar diagramas de clases, genera código desde diagramas. Proporciona abundantes tutoriales, demostraciones interactivas y proyectos de UML. Esta herramienta permite modelado colaborativo con Sistema de Versiones Concurrente (CVS) y Subversion, generación de bases de datos (transformación de diagramas entidad-relación en tablas de la base de datos), importación y exportación a ficheros XML, distribución automática de diagramas, entre otras características. (15)

1.7 Entorno de desarrollo: Eclipse

Se utiliza la Plataforma Eclipse como entorno de desarrollo integrado, el cual es multiplataforma y de código abierto, desarrollado por la Fundación Eclipse y liberado bajo la Licencia Pública de Eclipse (EPL, por sus siglas en inglés). Eclipse puede ser extendido mediante módulos (*plug-ins* en inglés), lo cual permite soportar herramientas para manipular variados tipos de contenidos como por ejemplo XML, HTML, C y GIF. También permite extenderse usando lenguajes de programación como C/C++ y mediante módulos libremente disponibles es posible añadir control de versiones mediante Subversion. (16)

Principales características:

- Multiplataforma (GNU/Linux, Solaris, Mac OS X, Windows).
- Soporte para distintas arquitecturas.
- Estructura de plugin que hace sencillo añadir nuevas características y funcionalidades.

[2] CASE: Computer Aided Software Engineering, Ingeniería de software Asistida por Computadora

CAPÍTULO 2: ANÁLISIS Y DISEÑO

En este capítulo se expone el flujo de trabajo Análisis y Diseño realizado para dar respuesta al problema planteado, y se describe el modelo conceptual donde se evidencian los conceptos fundamentales de la problemática para la solución del trabajo. Además se define el modelo de diseño para la réplica del módulo de Adquisición del SCADA Galba.

2.1 Presentación de la solución

En el nuevo esquema de redundancia se replican los bloques de los puntos recolectados desde los dispositivos de campo, la mayoría de los comandos enviados desde las consolas HMI y el estado de las comunicaciones con los dispositivos y subcanales. No se replican los comandos de sincronización y escritura de puntos, excepto los de escritura de un punto virtual manual. Con este esquema se eliminan las constantes copias de las memorias compartidas hacia el servidor redundante la cual solo se realizará al iniciar el servidor redundante.

El servidor redundante recibe las memorias compartidas al iniciar. Una vez que éstas se hayan recibido, se inician los procesos de puntos, alarmas y comandos. La recolección y publicación quedan detenidas hasta que este servidor promueva a principal.

Al llegar un bloque de puntos al servidor redundante se planifica una tarea para procesarlo siguiendo el mismo flujo de procesamiento que en el servidor principal. Los estados de las comunicaciones con los dispositivos y subcanales al llegar al servidor secundario son almacenados en memorias compartidas y se notifica al proceso de alarmas en caso de tenerlas configuradas. Si ambos servidores tienen un correcto funcionamiento de sus procesos las memorias compartidas estarán actualizadas y las salidas que muestren deben ser las mismas.

Con el objetivo de eliminar las inconsistencias que pueden ocurrir durante la sincronización total de las memorias compartidas, el servidor principal realiza una copia del estado de las regiones de las memorias compartidas que serán modificadas. Estas se envían al servidor redundante y reemplazan las regiones correspondientes, quedando eliminadas todas las inconsistencias entre los datos de las memorias en ambos servidores.

2.2 Modelo conceptual

“Se llama modelo del dominio o conceptual a la representación visual de los conceptos u objetos del mundo real en un dominio de interés. Es el mecanismo fundamental para comprender el dominio del problema y para establecer conceptos comunes. ” (17)

En la Figura 3 se presenta el modelo conceptual del módulo de Adquisición.

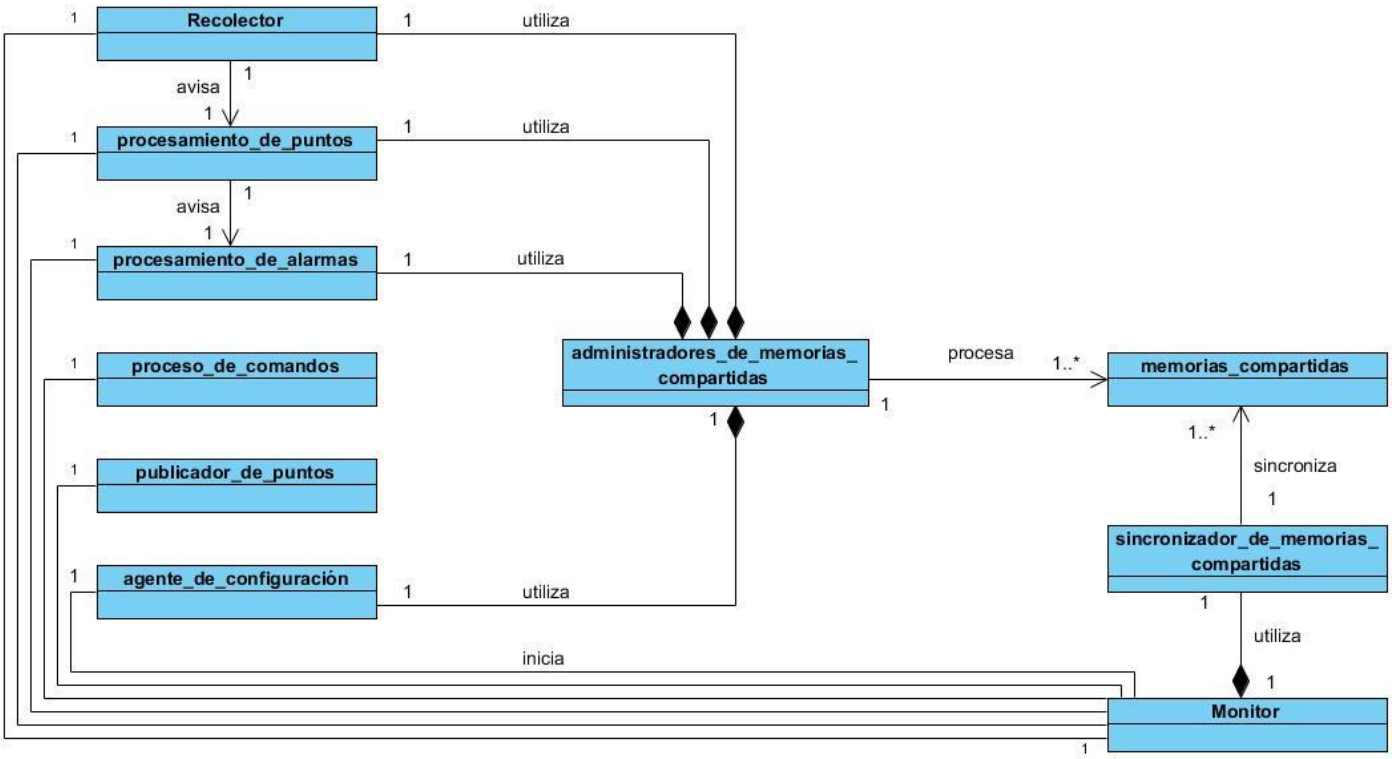

Figura 3: Diagrama del modelo conceptual del SCADA Galba.

En la Figura 4 se presenta el modelo conceptual de la solución.

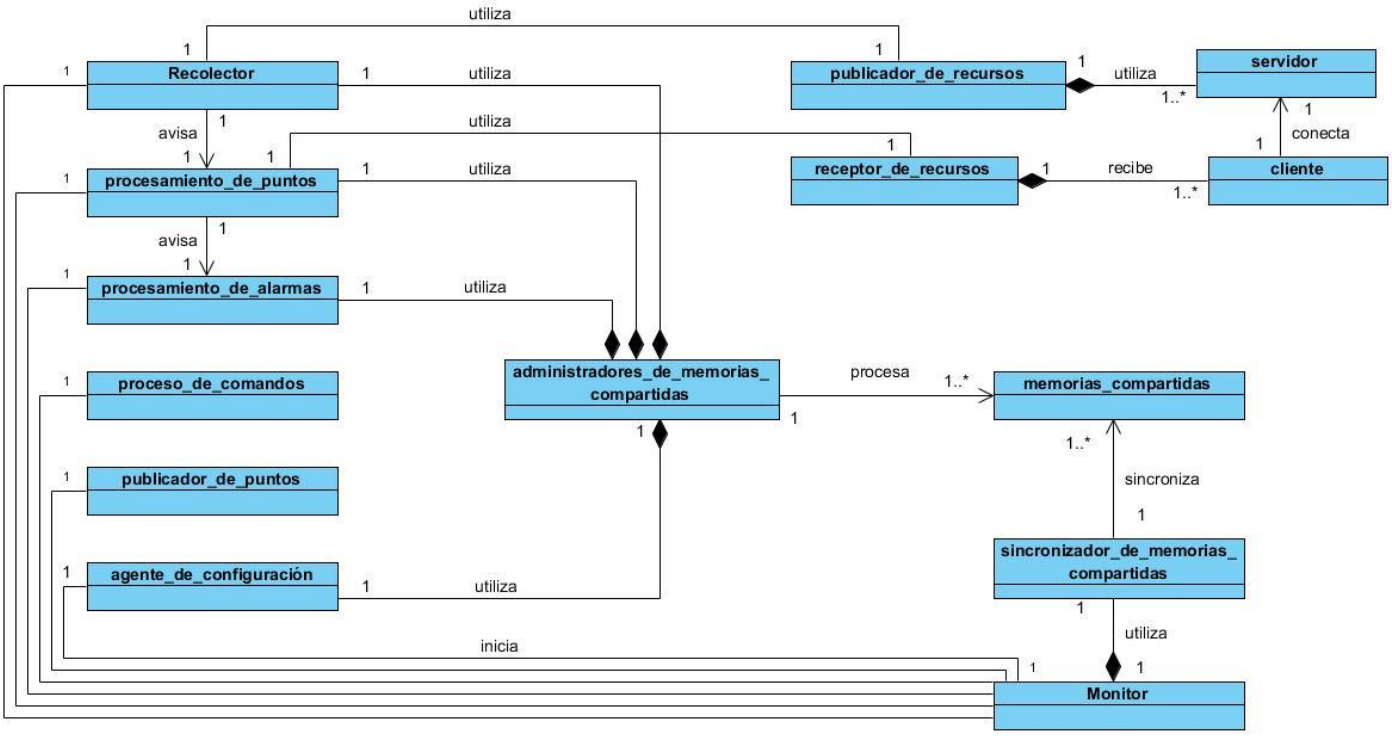

Figura 4: Diagrama del modelo conceptual de la solución propuesta.

A continuación se describen los elementos que integran el modelo de dominio de la solución propuesta para el módulo de Adquisición:

Recolector: es el responsable de adquirir la información del nivel de campo empleando una representación interna de los puntos agrupados según su frecuencia de lectura configurada, crea y planifica los bloques de variables a ser encuestados. Recolecta los puntos, los estados de las comunicaciones de los subcanales y dispositivos.

Procesamiento de puntos: recibe y procesa los bloques de puntos provenientes del recolector. Al recibir un mensaje del recolector notificando la existencia de puntos, accede a la memoria compartida del dispositivo y toma el bloque recolectado a procesar. Una vez que procese ese bloque, guarda los puntos en la memoria compartida de punto y avisa al procesamiento de alarmas la existencia de este. En cuanto al

servidor secundario este procesamiento se realiza con las mismas funciones que el servidor principal, teniendo en cuenta que recibirá los recursos del recolector del servidor principal.

Procesamiento de alarmas: es el responsable de analizar cada muestra recolectada para chequear la activación de las alarmas previamente configuradas en cada punto y almacena los resultados en su memoria compartida.

Proceso de comandos: a través de este se reciben comandos cuya ejecución implican a otros procesos del módulo de Adquisición y los envía al servidor de respaldo para ser procesados.

Publicador de puntos: es el publicador del módulo de Adquisición, accede a las memorias compartidas publicando a los clientes la actualización de los puntos, alarmas y eventos.

Administradores de memorias compartidas: es utilizado por los procesos antes descritos para procesar las memorias compartidas.

Sincronizador de memorias compartidas: realiza la réplica del servidor principal hacia el servidor redundante.

Monitor: realiza el monitoreo de los servicios que componen el sistema SCADA. Cada servidor tendrá ejecutándose este servicio de monitoreo. Él se encarga de verificar el estado de cada servicio del sistema, de manera directa a través de *sockets* TCP o UDP, o cualquier otro medio de comunicación.

Publicador de recursos: establece comunicación con el servidor redundante y se encarga de enviar los recursos del recolector hacia él. Este utiliza uno o varios **Servidores**, los cuales se encargan de aceptar las conexiones con el **Cliente** para enviar los datos.

Receptor de recursos: establece comunicación con el servidor principal y se encarga de recibir los recursos enviados desde el servidor principal. Este utiliza uno o varios **Clientes**, los cuales se encargan de conectarse al **Servidor** y recibir los datos.

2.3 Requisitos

2.3.1 Captura de requisitos

Para el proceso de captura de requisitos se utilizaron las técnicas:

Tormenta de ideas: Esta técnica es usada para generar nuevas ideas y encontrar la solución a cuestiones específicas. Es muy común en los comienzos del proceso de ingeniería de requisitos. Es una herramienta de trabajo grupal que facilita el surgimiento de nuevas ideas sobre un tema o problema determinado. La lluvia de ideas es una técnica de grupo para generar ideas originales. (18)

2.3.2 Requisitos funcionales

Un requisito funcional define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los requistos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas. (19)

En la siguiente tabla se exponen los requisitos funcionales especificados para la redundancia del módulo de Adquisición del SCADA Galba, los cuales son:

Tabla 1: Requisitos funcionales.

No.	Nombre de los requisitos funcionales
RF1	Replicar los comandos.
RF2	Replicar los bloques de puntos.
RF3	Replicar los estados de las comunicaciones de los dispositivos.
RF4	Replicar los estados de las comunicaciones de los subcanales.
RF5	Replicar las memorias compartidas.

2.3.3 Requisitos no funcionales

Los requisitos no funcionales complementan a los funcionales y las cualidades necesarias para que el producto sea eficaz. (20) Estos son requisitos que imponen restricciones en el diseño o la implementación. Son propiedades o cualidades que el producto debe tener, como por ejemplo: requisitos de apariencia, de usabilidad, de rendimiento y portabilidad, entre otros. (21)

Requisitos de diseño y de implementación

RNF1: El software se debe implementar usando el lenguaje de programación C++.

RNF2: Deben mantener la capacidad de funcionar en el sistema operativo Debian GNU/Linux en la versión 6.0.

RNF3: El código debe cumplir con los estándares de codificación establecidos por el Centro de Informática Industrial.

Requisitos del Hardware

RNF4: Como requerimientos mínimos que se necesitan para el esquema de la réplica se tienen:

- Memoria RAM de 1 GB.
- Disco duro de 160 GB.
- Microprocesador a 1.64 GHz.

Requisitos de rendimiento

RNF5: El tiempo que demora el servidor de respaldo en iniciar cuando el servidor principal falla debe ser menor de 40 segundos.

2.4 Historia de Usuario

Las historias de usuario (HU) son técnicas utilizadas para la representación de un requisito de software utilizando el lenguaje común del usuario. Algunas características de las HU es la participación del equipo en la toma de decisiones, proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo conllevará su implementación. Consiste en una descripción sencilla y clara de las actividades y acciones de los requisitos del sistema. (22)

A continuación se describen las historias de usuario que se tomaron para la modificación del esquema de la réplica:

Tabla 2: HU Replicar los comandos.

Historia de Usuario	
Número: HU1	**Nombre del requisito:** Replicar los comandos.
Programador: Yairilee Cruz Castillo.	**Iteración Asignada:** Primera
Prioridad: Alta.	**Tiempo Estimado:** Treinta días.
Riesgo en Desarrollo: Alta.	**Referencia:** HU5
	Tiempo Real: Cuatro semanas.
Descripción: Cuando el servidor principal reciba los comandos, se los envía al servidor redundante.	
Observaciones:	

Tabla 3: HU Replicar los bloques de puntos.

Historia de Usuario	
Número: HU2	**Nombre del requisito:** Replicar los bloques de puntos.
Programador: Yairilee Cruz Castillo.	**Iteración Asignada:** Primera
Prioridad: Alta.	**Tiempo Estimado:** Treinta días.
Riesgo en Desarrollo: Alta.	**Referencia:** HU5

	Tiempo Real: Cuatro semanas.
Descripción: Cuando el servidor principal reciba los bloques de puntos, se los envía al servidor redundante.	
Observaciones:	

Tabla 4: HU Replicar los estados de las comunicaciones de los dispositivos.

Historia de Usuario		
Número: HU3	**Nombre del requisito:** Replicar los estados de las comunicaciones de los dispositivos.	
Programador: Yairilee Cruz Castillo.		**Iteración Asignada:** Primera
Prioridad: Alta.		**Tiempo Estimado:** Treinta días.
Riesgo en Desarrollo: Alta.		**Referencia:** HU5
		Tiempo Real: Cuatro semanas.
Descripción: Cuando el servidor principal determine los estados de las comunicaciones de los dispositivos, se los envía al servidor redundante.		
Observaciones:		

Tabla 5: HU Replicar los estados de las comunicaciones de los subcanales.

Historia de Usuario	
Número: HU4	**Nombre del requisito:** Replicar los estados de las comunicaciones de los subcanales.
Programador: Yairilee Cruz Castillo.	**Iteración Asignada:** Primera
Prioridad: Alta.	**Tiempo Estimado:** Treinta días.
Riesgo en Desarrollo: Alta.	**Referencia:** HU5
	Tiempo Real: Cuatro semanas.
Descripción: Cuando el servidor principal determine los estados de las comunicaciones de los subcanales, se los envía al servidor redundante.	
Observaciones:	

Tabla 6: HU Replicar las memorias compartidas.

Historia de Usuario	
Número: HU5	**Nombre del requisito:** Replicar las memorias compartidas.
Programador: Yairilee Cruz Castillo.	**Iteración Asignada:** Primera
Prioridad: Alta.	**Tiempo Estimado:** Treinta días.
Riesgo en Desarrollo: Alta.	**Referencia:**-
	Tiempo Real: Cuatro semanas.

Descripción: Cuando el servidor redundante se ejecute, el servidor principal le envía las memorias compartidas.
Observaciones:

2.5 Modelo de Diseño

2.5.1 Diagrama de paquetes

El objetivo de los diagramas es obtener una visión más clara del sistema de información orientado a objetos, organizándolo en subsistema, agrupando los elementos del análisis, diseño o construcción y detallando las relaciones de dependencia entre ellos. (17)

En la Figura 5 se presenta el diagrama de paquetes de la solución.

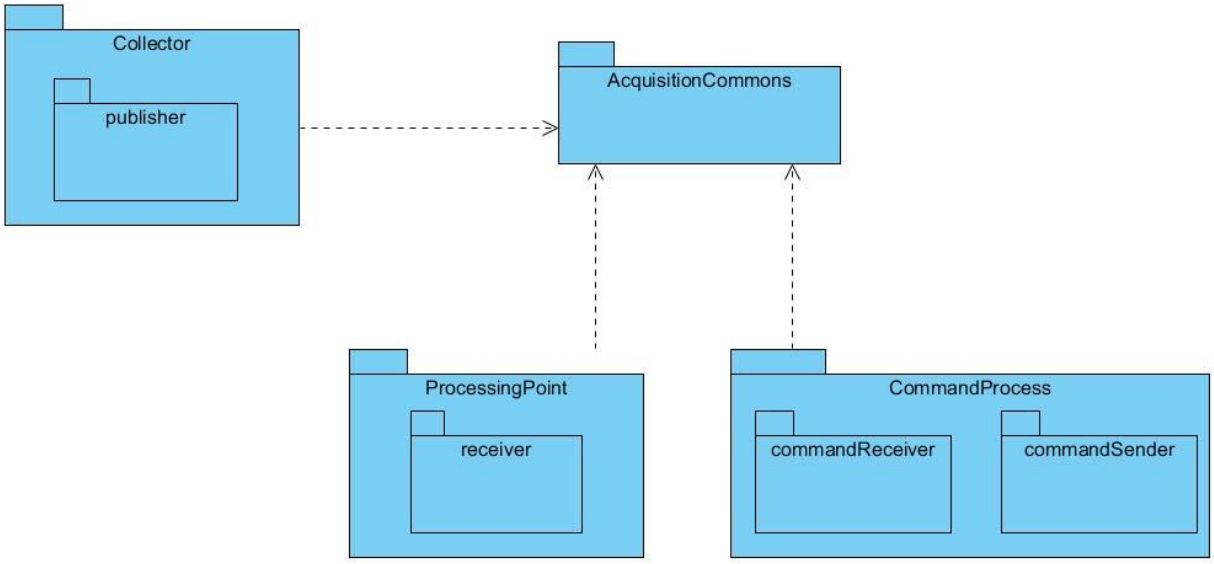

Figura 5: Diagrama de paquetes.

AcquisitionCommons: agrupa el conjunto de clases comunes al resto de los paquetes. **Collector:** Se adiciona el paquete *publisher*, donde se agrupan las clases utilizadas para el envío al servidor redundante de los bloques de puntos y los estados de las comunicaciones de los dispositivos y los subcanales.

ProcessingPoint: se adiciona el paquete *receiver*, donde se agrupan las clases utilizadas para recibir en

el servidor redundante los bloques de puntos y los estados de las comunicaciones de los dispositivos y los subcanales.

CommandProcess: se adiciona el paquete *commandSender*, el cual contiene la clase que envía los comandos al servidor redundante. Se adiciona el paquete *commandReceiver*, el cual agrupa las clases utilizadas para recibir los comandos en el servidor redundante.

2.5.2 Diagrama de clases

El diagrama de clases es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. (23)

El diagrama de clases del diseño se realiza en la disciplina análisis y diseño para utilizarlo como referencia en la disciplina implementación. A continuación en la Figura 6 se muestra el diagrama de clases del nuevo esquema de réplica.

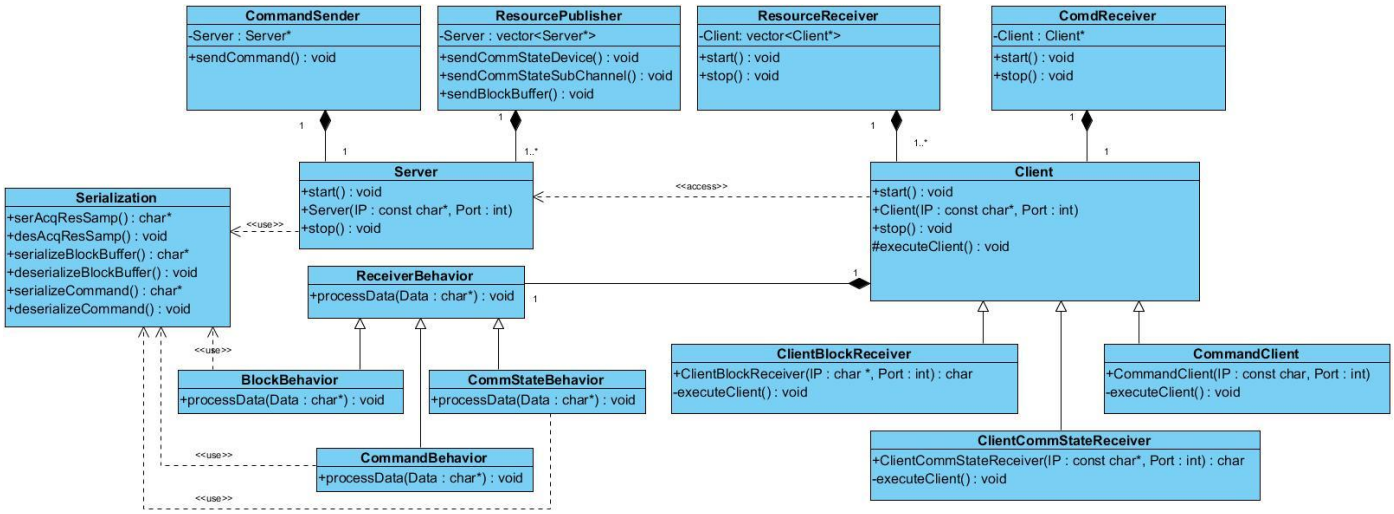

Figura 6: Diagrama de clases del diseño.

2.5.3 Descripción de clases y métodos del diseño

Tabla 7: Descripción de la clase: ResourcePublisher.

Descripción de la clase: *ResourcePublisher*	
Elemento	**Descripción**
ResourcePublisher	Envía los recursos del servidor principal al servidor redundante.
Métodos	
sendCommStateDevice(): void	Envía el estado de las comunicaciones de los dispositivos del servidor principal al servidor redundante.
sendCommStateSubChannel(): void	Envía el estado de las comunicaciones de los subcanales del servidor principal al servidor redundante.
sendBlockBuffer(): void	Envía los bloques de puntos del servidor principal al servidor redundante.

Tabla 8: Descripción de la clase: ResourceReceiver.

Descripción de la clase: *ResourceReceiver*	
Elemento	**Descripción**
ResourceReceiver	Esta clase tiene la responsabilidad de recibir los recursos enviados desde el servidor principal.
Métodos	
start(): void	Inicia el receptor de recursos para ejecutar clientes.
stop(): void	Termina la conexión con los clientes conectados al receptor de recursos.

Tabla 9: Descripción de la clase: Server.

Descripción de la clase: *Server*	
Elemento	**Descripción**
Server	Esta clase tiene la responsabilidad de aceptar la conexión con la clase Client y enviar los datos.
Métodos	
start(): void	Comienza la ejecución del servidor.
Server (IP: const char*, Port: int): char	Construye las instancias de la clase.
stop(): void	Termina la conexión con los clientes conectados al receptor de recursos.

Tabla 10: Descripción de la clase: Client.

Descripción de la clase: *Client*	
Elemento	**Descripción**
Client	Esta clase tiene la responsabilidad de conectarse al Server y recibir los datos.
Métodos	
start(): void	Comienza la ejecución del Cliente.
stop (): void	Termina la conexión con los clientes conectados.
Client (IP: const char*, Port: int): char	Construye las instancias de la clase.

Tabla 11: Descripción de la clase: Serialization.

Descripción de la clase: *Serialization*	
Elemento	**Descripción**
Serialization	Serializa los recursos que se envían al servidor redundante.
Métodos	
serAcqResSamp (): char*	Serializa el estado de las comunicaciones de los dispositivos y subcanales.
desAcqResSamp (): void	Deserializa el estado de las comunicaciones de los dispositivos y subcanales.
serializeBlockBuffer(): char*	Serializa los bloques de puntos.
deserializeBlockBuffer(): void	Deserializa los bloques de puntos.
serializeCommand(): char*	Serializa los comandos.
deserializeCommand(): void	Deserializa los comandos.

Tabla 12: Descripción de la clase: CommandSender.

Descripción de la clase: *CommandSender*	
Elemento	**Descripción**
CommandSender	Esta clase tiene la responsabilidad de enviar los comando al servidor redundante.
Métodos	
sendCommand (): void	Envía un comando al servidor redundante.

executeClient (): void	Se conecta con el servidor principal.

Tabla 13: Descripción de la clase: CommStateBehavior.

Descripción de la clase: *CommStateBehavior*	
Elemento	**Descripción**
CommStateBehavior	Representa el comportamiento del estado de las comunicaciones con subcanales y dispositivos.
Métodos	
processData(char* data) (): void	Procesa el estado de las comunicaciones de los subcanales y dispositivos.

Tabla 14: Descripción de la clase: ClientCommStateReceiver.

Descripción de la clase: *ClientCommStateReceiver*	
Elemento	**Descripción**
ClientCommStateReceiver	Recibe el estado de las comunicación de los dispositivos y subcanales desde el servidor principal.
Métodos	
ClientCommStateReceiver(IP: const char*, Port: int): char	Construye las instancias de la clase.
executeClient(): void	Recibe el estado de las comunicaciones de los dispositivos y subcanales desde el servidor principal y notifica al Proceso de alarmas para su procesamiento.

Tabla 15: Descripción de la clase: ClientBlockReceiver.

Descripción de la clase: *ClientBlockReceiver*	
Elemento	**Descripción**
ClientBlockReceiver	Recibe los bloques desde el servidor principal.
Métodos	
ClientBlockReceiver (IP: const char*, Port: int): char	Construye las instancias de la clase.
executeClient(): void	Recibe los bloques de puntos desde el servidor principal y los procesa.

Tabla 16: Descripción de la clase: BlockBehavior.

Descripción de la clase: *BlockBehavior*	
Elemento	**Descripción**
BlockBehavior	Representa el comportamiento de los bloques de puntos.
Métodos	
processData(char*): void	Procesa los bloques de puntos.

Tabla 17: Descripción de la clase: ComdReceiver.

Descripción de la clase: *ComdReceiver*	
Elemento	**Descripción**
ComdReceiver	Esta clase crea e inicia los clientes para recibir los comandos.

Métodos	
start(): void	Inicia la recepción de los comandos.
stop(): void	Termina la conexión.

Tabla 18: Descripción de la clase: CommandClient.

Descripción de la clase: *CommandClient*	
Elemento	**Descripción**
CommandClient	Recibe los comandos desde el servidor principal.
Métodos	
CommandClient (IP: const char*, Port: int): char	Construye las instancias de la clase.
executeClient(): void	Recibe los comandos desde el servidor principal y los procesa.

Tabla 19: Descripción de la clase: CommandBehavior.

Descripción de la clase: *CommandBehavior*	
Elemento	**Descripción**
CommandBehavior	Representa el comportamiento de los comandos.
Métodos	
processData(char*): void	Procesa los comandos.

Tabla 20: Descripción de la clase: ReceiverBehavior.

Descripción de la clase: *ReceiverBehavior*	
Elemento	**Descripción**
ReceiverBehavior	Encapsulan los comportamientos de los recursos que se reciben.
Métodos	
processData(char*): void	Procesa los recursos que recibe.

2.5.4 Patrones de diseño

Los patrones de diseño se han convertido en una técnica importante para el uso del conocimiento de software. Cada patrón provee información sobre su diseño, describiendo las clases, métodos y relaciones que resuelven un problema de diseño en particular. (24)

Patrones GRASP.

Los patrones GRASP (acrónimo de General Responsibility Assignment Software Patterns), son patrones generales de software para asignación de responsabilidades. GRASP da principios generales para asignar responsabilidades, y se utiliza sobre todo en la realización de diagramas de interacción, aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software. (24)

Los patrones utilizados son:

El GRASP de **experto** en información es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda información necesaria para crearlo. Se utiliza en las clases **Client** y **Server,** donde **Client** tiene la responsabilidad de conectarse al **Server** y recibir los datos; por otro lado el **Server** tiene la responsabilidad de aceptar la conexión con el **Client** y enviar los datos.

Creador se utiliza para a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Se utiliza en las clases **ResourcePublisher** y **ResourceReceiver**.

Patrones GoF.

Entre sus principales características están:

- Son soluciones concretas. Proponen soluciones a problemas concretos, no son teorías genéricas.
- Son soluciones técnicas. Indican resoluciones técnicas basadas en Programación Orientada a Objetos (POO).
- Favorecen la reutilización de código. Ayudan a construir software basado en la reutilización, a construir clases reutilizables. Los propios patrones se reutilizan cada vez que se vuelven a aplicar. (17)

El patrón Singleton: Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella. El patrón de diseño *singleton* (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. (25) Este se utiliza en la clase **ResourcePublisher.**

El patrón Estrategia: Se clasifica como patrón de comportamiento porque determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. Este se utiliza en la clase **ReceiverBehavior**, la cual hereda los comportamientos específicos, como el estado de las comunicaciones de los dispositivos, de los subcanales, bloques y comandos.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

Introducción

En el presente capítulo se abordan los aspectos relacionados con la implementación y prueba de la solución propuesta, así como la evaluación de la calidad del diseño de clases mediante la utilización de métricas.

3.1 Implementación

3.1.1 Estándares de codificación

Los estándares de codificación (llamados estilos) tienen como objetivo fomentar el uso de buenas prácticas de programación. En general, un estándar de codificación son reglas que se siguen para la escritura del código fuente. De tal manera que otros programadores se les facilite entender el código (como identificar las variables, las funciones o métodos).

El estilo de código utilizado para integrar la solución es el que establece el Centro de Informática Industrial, registrado en el informe 0120_1 Estándares de codificación para C++ del Programa de Mejora del Proyecto SCADA Guardián del Alba.

3.2 Métricas para evaluar el diseño propuesto

IEEE[3] define métrica como “una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado”. (26)

Las métricas se centran en cuantificar tanto la complejidad, como la funcionalidad y eficiencia inmersa en el desarrollo de software. Inclina sus objetivos a mejorar la comprensión de la calidad del producto, a estimar la efectividad del proceso y mejorar la calidad del trabajo. (27)

[3] IEEE de las siglas en inglés **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers (Instituto de Ingeniería Eléctrica y Electrónica).

3.2.1 Métricas propuestas por Lorenz y Kidd

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

- **Responsabilidad:** consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación**: consiste en el grado de dificultad que tiene implementado un diseño de clases determinado.
- **Reutilización:** consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento:** consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras. Está muy ligada a la característica de Reutilización.
- **Complejidad del mantenimiento:** consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software.
- **Cantidad de pruebas:** consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto diseñado.

Se evaluó el diseño mediante las métricas **Tamaño Operacional de Clases (TOC)** y **Relaciones entre Clases (RC)** que fueron propuestas por Lorenz y Kidd. A continuación se describen sus funcionalidades. (27)

- **Tamaño Operacional de Clases:** está dado por el número de métodos asignados a una clase.

La Tabla 21 describe los atributos que se evalúan al aplicar la métrica TOC.

Tabla 21: Tamaño operacional de clases (TOC).

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución del grado de reutilización de la clase.

- **Relaciones entre Clases:** está dado por el número de relaciones de uso de una clase con otra.

La Tabla 22 describe los atributos de calidad que se miden al evaluar la métrica RC.

Tabla 22: Relaciones entre clases (RC).

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del acoplamiento de la clase.
Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

3.2.2 Resultados de la métrica TOC.

La Tabla 23 muestra el cálculo llevado a los atributos de la métrica.

Tabla 23: Cálculo de los atributos de calidad de la métrica TOC.

	Categoría	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

La Figura 7 refleja que la mayoría de las clases tienen pocos procedimientos y que el funcionamiento general del sistema está distribuido entre las diferentes clases.

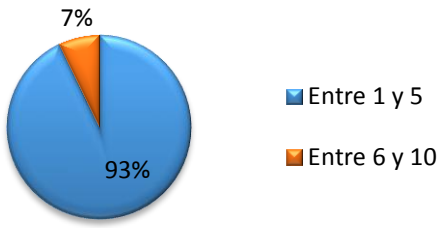

Figura 7: Representación de la métrica TOC.

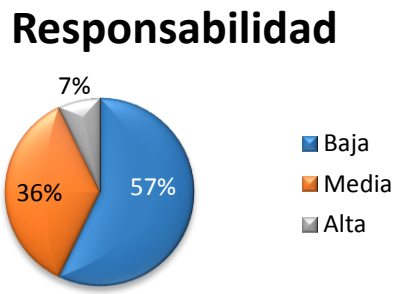

Figura 8: Resultado de la responsabilidad aplicando la métrica TOC.

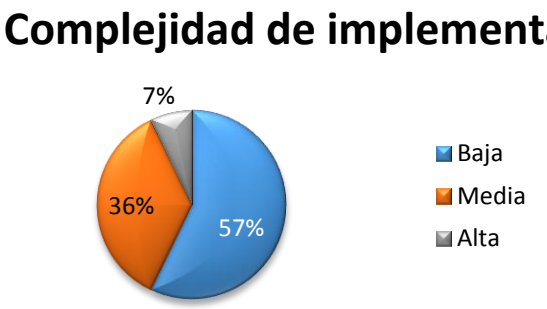

Figura 9: Resultado de la complejidad de implementación aplicando la métrica TOC.

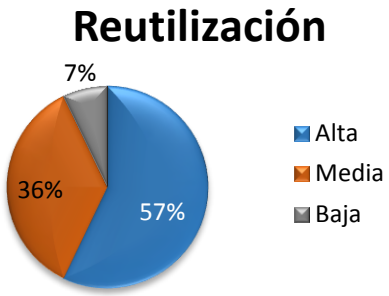

Figura 10: Resultado de la reutilización aplicando la métrica TOC.

Haciendo un análisis de los resultados obtenidos (ver Figuras 8, 9 y 10) de la métrica TOC, se puede concluir que el diseño de clases tiene una calidad aceptable ya que la mayoría tienen una baja responsabilidad, un alto grado de reutilización y son relativamente fáciles de implementar, permitiendo reducir el tiempo empleado para el soporte y garantizar que los fallos que puedan ocasionarse en el funcionamiento de una clase no tengan un gran impacto en el sistema.

3.2.3 Resultados de la métrica RC.

La Tabla 24 muestra el cálculo llevado a los atributos de la métrica.

Tabla 24: Cálculo de los atributos de calidad de la métrica RC.

	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2

	Categoría	Criterio
Complejidad Mant.	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

	Categoría	Criterio
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.

	Categoría	Criterio
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

La Figura 11 refleja que la mayoría de las clases tienen pocas dependencias.

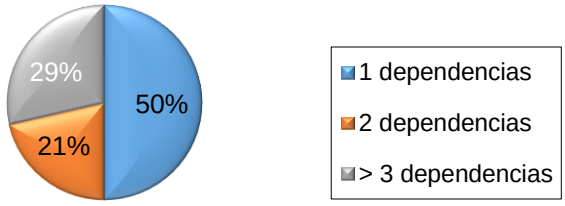

Figura 11: Representación de la métrica RC.

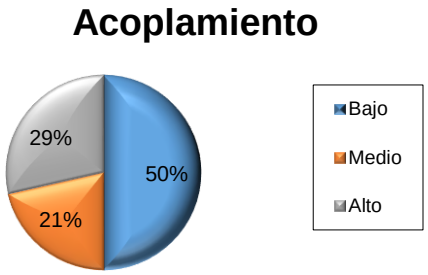

Figura 12: Resultado del acoplamiento aplicando la métrica RC.

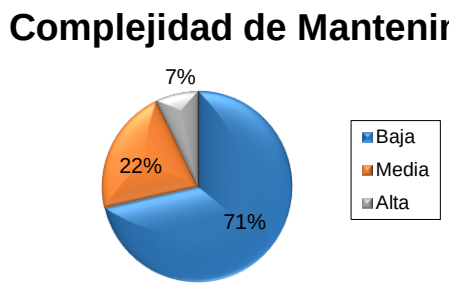

Figura 13: Resultado de la complejidad de mantenimiento aplicando la métrica RC.

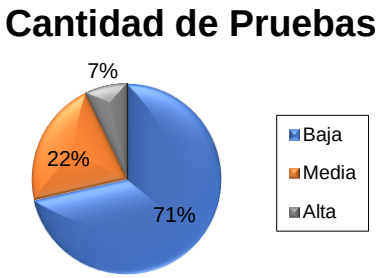

Figura 14: Resultado de la cantidad de pruebas aplicando la métrica RC.

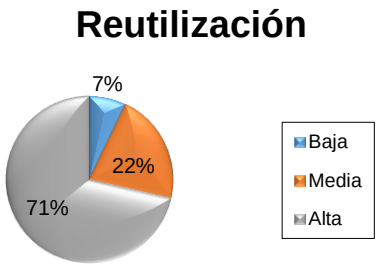

Figura 15: Resultado de la reutilización aplicando la métrica RC.

Haciendo un análisis de los resultados obtenidos (ver Figuras 12, 13, 14 y 15) de la métrica RC, se puede concluir que el diseño de clases tiene una calidad aceptable ya que la mayoría requieren de pocas pruebas porque presentan una baja complejidad para el mantenimiento y un alto grado de reutilización, permitiendo

reducir el tiempo empleado para el soporte. Además, la mitad de las clases cuentan con un bajo acoplamiento.

3.3 Pruebas de software

Una vez culminado con la implementación del sistema, es de vital importancia realizarle pruebas para comprobar su correcto funcionamiento. Entre sus objetivos está verificar la integración adecuada de los componentes, verificar que todos los requisitos se hayan implementado correctamente, y asegurar que los defectos encontrados se hayan corregido. (28)

3.3.1 Pruebas de Rendimiento

Las pruebas de rendimiento son utilizadas con frecuencia para verificar y validar que un sistema cumple con los requisitos de rendimiento. Pueden ser empleadas para comparar dos sistemas en cuanto a su rendimiento o detectar los problemas que degradan el rendimiento de un sistema. (17)

Se realizaron pruebas de rendimiento con el objetivo de comparar el esquema de réplica en la versión Miranda R2 y el nuevo esquema de réplica implementado. En la siguiente tabla se muestran los tiempos de desactualización que existen en ambas variantes.

Tabla 25: Tiempo de desactualización del módulo de Adquisición.

Cantidad de Puntos	Esquema de réplica de Miranda R2 (milisegundos)	Nuevo esquema de réplica (milisegundos)
1500	**12452**	**1**
15000	**14265**	**12**
25000	**15334**	**38**

El proceso de la réplica se realizó en el siguiente ambiente de prueba:

- Dos ordenadores con 1 GB de memoria RAM, 1 TB de disco duro y procesador Intel(R) Core(TM) i3-2120 CPU 3.30 GHz, conectados en una misma red LAN sin utilizar conexión punto a punto.

- Todos los puntos fueron configurados en un solo dispositivo, utilizando el protocolo de comunicación ModbusTCP.

Según la Tabla 25, se concluye que el nuevo esquema de réplica disminuye significativamente el tiempo de la desactualización, logrando un factor de reducción superior a 400 respecto a la versión Miranda R2. Además, se logró que el tiempo de la desactualización del servidor de respaldo sea inferior a un segundo.

3.3.2 Pruebas de Caja Blanca

La prueba de caja blanca, denominada a veces prueba de caja de cristal, es un método de diseño de casos de prueba que usa la estructura del control de diseño procedimental para obtener los casos de prueba. (27)

Mediante la prueba de la caja blanca el ingeniero del software puede obtener casos de prueba que:

- Ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejerciten todas las decisiones lógicas en las vertientes verdaderas y falsas.
- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

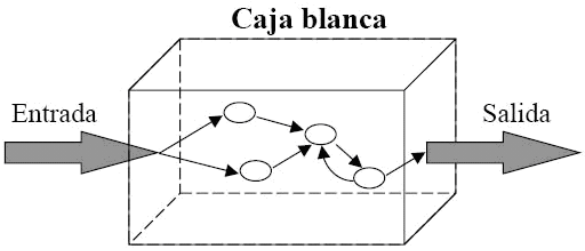

Figura 16: Representación de pruebas de caja blanca.

La prueba del camino básico es una técnica de prueba de la caja blanca, que permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez. (29) A continuación la Figura 17 enumera las sentencias de código del método **sendData()**.

```
bool Server::sendData(char * buff, size_t size)
{
   if ( buff != 0 and size > 0 ) //1
   {
      if ( sendInfo(buff, size) > 0 ) //2
      {
         return true; //3
      }
      else //4
      {
         acceptClients(); //5
         return false;    //5
      }
   }
   else //6
   {
      return false; //7
   }
} //8
```

Figura 17: Código fuente del método sendData().

La Figura 18 muestra el grafo del flujo asociado al método **sendData()**.

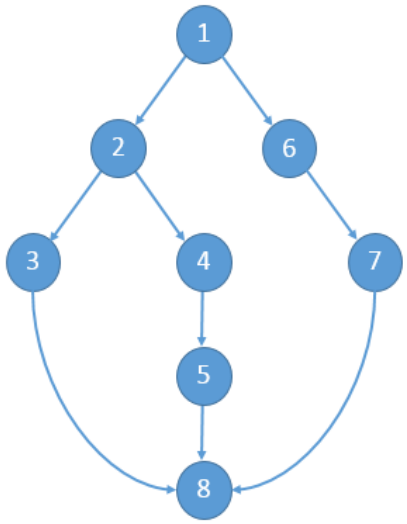

Figura 18: Grafo de flujo asociado al método sendData().

La complejidad ciclomática es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y proporciona un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Después de haberse realizado el grafo, se calcula la complejidad ciclomática por tres fórmulas distintas, las cuales deben dar el mismo resultado para comprobar que el cálculo sea correcto.

La complejidad ciclomática, V (G) se define como:

1. **V (G) = R** donde: **R** representa la cantidad total de regiones del grafo.
   V (G) = 3

2. **V (G) = A – N + 2** donde: **A** es el número de aristas del grafo y **N** es el número de nodos.
   V (G) = 9 – 8 + 2
   V (G) = 3

3. **V (G) = P + 1** donde: **P** es el número de nodos predicados contenidos en el grafo.
   V (G) = 2 + 1
   V (G) = 3

El cálculo de las diferentes fórmulas anteriormente nombradas arrojó el mismo resultado, por esto se puede afirmar que la complejidad ciclomática del método es tres. Esto significa que existen tres posibles caminos por donde el flujo puede circular. Este valor representa el número mínimo de casos de prueba para el procedimiento tratado.

- **Camino básico #1: 1 – 2 – 3 – 8**
- **Camino básico #2: 1 – 2 – 4 – 5 – 8**
- **Camino básico #3: 1 – 6 – 7 – 8**

Luego de tener elaborados los grafos de flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

Caso de prueba para el camino básico # 1
Camino: 1 – 2 – 3 – 8
Descripción: Los datos se obtienen a través del parámetro buff y parámetro size.
Entrada: buff= bloques de puntos, size=1000.
Resultados esperados: Se envían los datos almacenados en la variable buff, no se aceptan los clientes y se devuelve valor verdadero.
Resultados obtenidos: Satisfactorio.

Caso de prueba para el camino básico # 2
Camino: 1 – 2 – 4 – 5 – 8
Descripción: Los datos se obtienen a través del parámetro buff y parámetro size.
Entrada: buff= bloques de puntos, size=1000.
Resultados esperados: No se envían los datos almacenados en la variable buff, se aceptan los clientes y se devuelve valor falso.
Resultados obtenidos: Satisfactorio.

Caso de prueba para el camino básico # 3
Camino: 1 – 6 – 7 – 8
Descripción: Los datos se obtienen a través del parámetro buff y parámetro size.
Entrada: buff= bloques de puntos, size=0.
Resultados esperados: No se envían los datos almacenados en la variable buff, no se aceptan los clientes y se devuelve valor falso.
Resultados obtenidos: Satisfactorio.

Con la realización de las pruebas de caja blanca al método **sendData()**, se obtuvieron los resultados esperados, debido a que las tres fórmulas explicadas arrojaron el mismo resultado y se cumplió la ejecución de cada camino básico al menos una vez con los parámetros de entrada y las salidas esperadas.

3.3.3 Pruebas de Caja Negra

La prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del modelo, fundamentalmente del sistema, sin tener mucho en cuenta la estructura interna del software. (30)

Descripción de los casos de prueba

A continuación se detallan los casos de prueba elaborados por cada uno de los requisitos funcionales para comprobar el correcto funcionamiento del esquema de la réplica:

Tabla 26: DCP Replicar los comandos.

Escenario	Descripción de la funcionalidad	Respuesta del sistema	Flujo central
Replicar los comandos.	Enviar los comandos del servidor principal al servidor redundante.	El servidor redundante notifica la llegada de los comandos.	EL proceso de procesamiento de comandos del servidor principal recibe los comandos del módulo HMI. El proceso de procesamiento de comandos del servidor principal envía los comandos al servidor redundante. El proceso de procesamiento de comandos del servidor de respaldo recibe los comandos y los procesa.

Tabla 27: DCP Replicar bloques de puntos.

Escenario	Descripción de la funcionalidad	Respuesta del sistema	Flujo central
Replicar bloques de puntos.	Enviar los bloques de puntos del servidor principal al servidor redundante.	El servidor redundante notifica la llegada del bloque de puntos.	El recolector del servidor principal recibe los bloques de puntos de los dispositivos. El recolector del servidor principal envía los bloques de puntos al servidor redundante. El proceso de procesamiento de puntos del servidor de respaldo recibe los bloques de puntos y los procesa.

Tabla 28: DCP Replicar las memorias compartidas.

Escenario	Descripción de la funcionalidad	Respuesta del sistema	Flujo central
Replicar las memorias compartidas.	Enviar las memorias compartidas del servidor principal al servidor redundante.	El servidor redundante notifica la llegada de las memorias compartidas.	Ejecución del servidor de respaldo. El servidor principal registra la información que existía en las regiones en las memorias compartidas que pueden ser modificadas. El servidor principal copia las memorias compartidas al servidor secundario. El servidor principal envía al servidor de respaldo el registro de la información de las regiones de las memorias compartidas.

			El servidor de respaldo utiliza el registro recibido para modificar la información de las memorias compartidas.

Tabla 29: DCP Replicar los estados de las comunicaciones de los dispositivos.

Escenario	**Descripción de la funcionalidad**	**Respuesta del sistema**	**Flujo central**
Replicar los estados de las comunicaciones de los dispositivos.	Enviar los estados de las comunicaciones de los dispositivos del servidor principal al servidor redundante.	El servidor redundante notifica la llegada de los estados de las comunicaciones de los dispositivos.	El proceso recolector del servidor principal determina los estados de las comunicaciones de los dispositivos. El proceso recolector del servidor principal envía los estados de las comunicaciones de los dispositivos al servidor de respaldo. El procesamiento de alarmas del servidor de respaldo recibe los estados de las comunicaciones de los dispositivos. El procesamiento de alarmas del servidor de respaldo procesa los estados de las comunicaciones de los dispositivos.

Tabla 30: DCP Replicar los estados de las comunicaciones de los subcanales.

Escenario	**Descripción de la funcionalidad**	**Respuesta del sistema**	**Flujo central**
Replicar los estados de las comunicaciones de los subcanales.	Enviar los estados de las comunicaciones de los subcanales del servidor principal al servidor redundante.	El servidor redundante notifica la llegada de los estados de las comunicaciones de los subcanales.	El proceso recolector del servidor principal determina los estados de las comunicaciones de los subcanales. El proceso recolector del servidor principal envía los estados de las comunicaciones de los subcanales al servidor de respaldo. El procesamiento de alarmas del servidor de respaldo recibe los estados de las comunicaciones de los subcanales. El procesamiento de alarmas del servidor de respaldo procesa los estados de las comunicaciones de los subcanales.

Una vez realizados los casos de prueba, se recogen las no conformidades detectadas. Para verificar la calidad del esquema de la réplica se realizaron tres iteraciones, las cuales arrojaron varias no conformidades. Estas se muestran en la Figura 19:

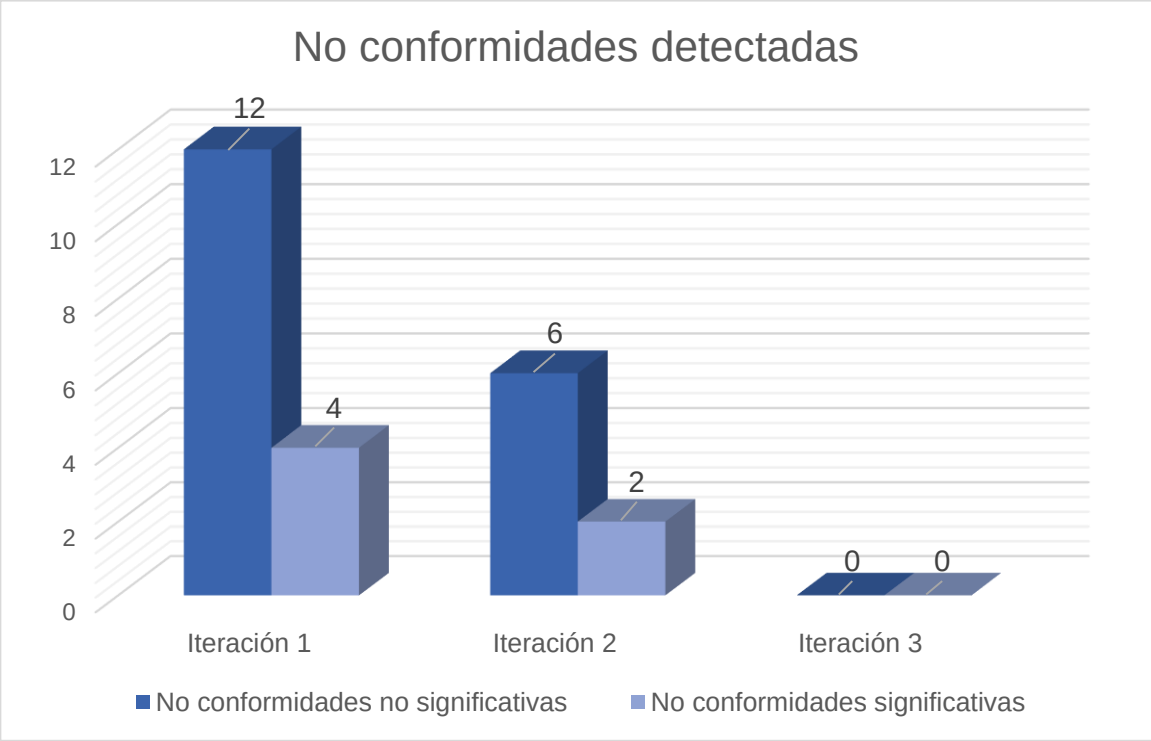

Figura 19: No conformidades detectadas en el nuevo esquema de la réplica.

CONCLUSIONES

Al término de la investigación se concluye que:

- El estudio realizado de los esquemas de réplica permitió crear un nuevo esquema para la redundancia del módulo de Adquisición.
- La utilización de patrones de diseño en el diagrama de clases propuesto permitió la reutilización de código y una baja complejidad de mantenimiento, lo cual quedó demostrado mediante las métricas de validación de diseño TOC y RC.
- Las pruebas realizadas demostraron el correcto funcionamiento del esquema de réplica propuesto y la reducción del tiempo de desactualización del servidor de respaldo, dándole así solución al problema planteado.

RECOMENDACIONES

En aras de extender el alcance del presente trabajo se recomienda:

- Evaluar la utilidad de aplicar algoritmos de compresión sin pérdida sobre la información a enviar al servidor redundante.
- Persistir en el servidor redundante la información de las memorias compartidas para evitar su pérdida ante reinicio, eliminándose en los casos posibles la réplica de las memorias compartidas.
- Implementar mecanismo para evitar la réplica de las memorias compartidas en caso que el servidor redundante pierda la conexión con el servidor principal por poco tiempo.

REFERENCIA BIBLIOGRÁFICA

1. San-José, Pablo Pérez y Eduardo Álvarez Alonso. *Estudio sobre la seguridad de los sistemas de monitorización y control de procesos e infraestructuras (SCADA).* s.l. : Instituto Nacional de Tecnologías de la Comunicación (INTECO), 2012.

2. *Sistema Supervisor Guardián del ALBA.Introducción a la Arquitectura del Guardian del ALBA.* Moisés Herrera Vázquez, Yaneisy Hernández y Jaime Fardales Pérez. 2008.

3. *Versión Miranda R2 del módulo de Adquisición del SCADA "Guardián del Alba".* Fariñas., Hebert Hernández. Cuba : s.n., 2013.

4. *Sistema Supervisor Guardián del ALBA. Base de Datos de Tiempo Real en el "GUARDIAN DEL ALBA".* Cuba : s.n., 2008.

5. Taylor, Zachary. *Designing High Availability Systems.* 2013.

6. Sanmiguel, Alfred Gutiérrez. *Clúster de alta disponibilidad y balanceo de carga sobre un servidor web.* 2012.

7. *SISTEMA PARA EL INTERCAMBIO SEGURO DE DATOS EN EL SCADA "GUARDIÁN DEL ALBA".* Daisy Diana Vargas Vento, David Vargas Henández. 2012.

8. *MySQL.* [En línea] Oracle, 2015. https://www.mysql.com/.

9. *PostgreSQL.* [En línea] The PostgreSQL Global Development Group, 2015. http://www.postgresql.org/docs/9.0/interactive/high-availability.html.

10. *Slony-I enterprise-level replication system.* [En línea] Command Prompt, Inc., 2010. http://www.slony.info/.

11. Sommerville, Ian. *Ingeniería del Software.* s.l. : Addison-Wesley, 2005.

12. Sánchez, Tamara Rodríguez. Metodología de desarrollo para la Actividad productiva de la UCI. Cuba : s.n., 2014.

13. UML. *UML.* [En línea] [Citado el: 17 de Enero de 2015.] www.uml.org.

14. Puerto., Sorio. *Lenguaje de Programación C++.* 2010.

15. Paradigm, Visual. [En línea] 2014. [Citado el: 17 de Enero de 2015.] www.visual-paradigm.com.

16. *eclipse.* [En línea] The Eclipse Foundation, 2015. [Citado el: 12 de febrero de 2015.] http://www.eclipse.org/.

17. Larman, Craig. *UML y Patrones.* 2004.

18. Ganesh, Gunda Sai. *Requirements Engineering: Elicitation Techniques.* University West, Departament of Technology, Mathematics and Computer Science : s.n., 2008.

19. Thayer, R.H y Merlin, D. *IEEE Software Requirement Engineering Second Edition. .* New York. : s.n., 1997.

20. PMBOK. *Guía de los Fundamentos para la Dirección de Proyectos 5ta Edicion.* 2013.

21. Stellman, Andrew y Greene, Jennifer. *Applied Software Project Management.* EEUU : O´Really Media., 2005.

22. PMOinformatica. La oficina de proyectos de informática. [En línea] 2014. http://www.pmoinformatica.com/2013/04/que-son-las-historias-de-usuario-7.html.

23. Cillero, Manuel. manuel.cillero. [En línea] 2014. http://manuel.cillero.es/doc/metrica-3/tecnicas/diagrama-de-clases.

24. Larman, Craig. *Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 2003.

25. Bates., Eric Freeman & Elisabeth Freeman with & Kathy Sierra & Bert. *O'Reilly -Head First Design Patterns.* 2005.

26. IEEE. *Introducción a la Ingeniería de Requisitos.* 2007.

27. Pressman. *Ingeniería de Software: Un enfoque práctico.* Sexta edición. 2005.

28. Capote García, Tayché. Conceptualización e implantación de un Laboratorio Industrial de Pruebas de Software. *Tesis de Maestría.* La Habana : s.n., 2011.

30. Echeverría Perez, Delvis. Desarrollo de una ontología de apoyo al procedimiento del Departamento de Pruebas de Software. *Tesis de Maestría.* La Habana, Universidad de las Ciencias Informáticas. : s.n., 2011.

ANEXOS

ANEXO. Instrumento de evaluación de las métricas TOC y RC.

Tabla 31: Instrumento de evaluación de la métrica TOC.

Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
ResourcePublisher	3	Media	Media	Media
BlockBehavior	1	Baja	Baja	Alta
ClientBlockReceiver	1	Baja	Baja	Alta
ClientCommStateReceiver	1	Baja	Baja	Alta
CommStateReceiver	1	Baja	Baja	Alta
ReceiverBehavior	1	Baja	Baja	Alta
ResourceReceiver	2	Media	Media	Media
Client	2	Media	Media	Media
Server	2	Media	Media	Media
Serialization	6	Alta	Alta	Baja
CommandClient	1	Baja	Baja	Alta
ComdReceiver	2	Media	Media	Media
CommandBehavior	1	Baja	Baja	Alta
CommandSender	1	Baja	Baja	Alta

Tabla 32: Instrumento de evaluación de la métrica RC.

Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
ResourcePublisher	2	Medio	Baja	Alta	Baja
BlockBehavior	1	Bajo	Baja	Alta	Baja
ClientBlockReceiver	1	Bajo	Baja	Alta	Baja
ClientCommStateReceiver	1	Bajo	Baja	Alta	Baja
CommStateReceiver	1	Bajo	Baja	Alta	Baja
ReceiverBehavior	4	Alto	Media	Media	Media
ResourceReceiver	2	Medio	Baja	Alta	Baja
Client	7	Alto	Alta	Baja	Alta
Server	3	Alto	Media	Media	Media
Serialization	4	Alto	Media	Media	Media
CommandClient	1	Bajo	Baja	Alta	Baja
ComdReceiver	2	Medio	Baja	Alta	Baja
CommandBehavior	1	Bajo	Baja	Alta	Baja
CommandSender	2	Medio	Baja	Alta	Baja