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

Título: "Mecanismo para la generación de trazas a partir de los cambios ocurridos en la configuración del SCADA Galba".

Autor:

Jorge Alfonso Orama Pérez

Tutores:

Ing. Miguel Ángel Socorro Borges

Ing. Julio Cesar Espronceda Pérez

La Habana, Cuba

Junio de 2015

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo en su beneficio.

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

_____

Firma del autor

Jorge Alfonso Orama Pérez

_____

Firma del tutor

Ing. Miguel Ángel Socorro Borges

_____

Firma del tutor

Ing. Julio Cesar Espronceda Pérez

Datos de contacto

Ing. Miguel Ángel Socorro Borges.

Graduado de: Ingeniero en Ciencias Informáticas.

Años de experiencia: 4 años.

Teléfono de contacto: 7 837 2317

E-mail: miguelsb@uci.cu

Tutor: Julio César Espronceda Pérez.

Graduado de: Ingeniero en Ciencias Informáticas.

Años de experiencia: 2 años.

Teléfono de contacto: 7 837 2745

E-mail: jcespronceda@uci.cu

Agradecimientos

Quisiera agradecer a todas aquellas personas, que en sentido general me ayudaron en la realización de la presente investigación.

A mis tutores:

Gracias por ser, además de tutores, amigos, por apoyarme siempre que los necesité, por no tener peros a la hora de aclararme una duda, gracias por ayudarme a hacerme ingeniero.

A mi novia:

Gracias, por estar siempre para mi, por preocuparte incluso más que yo mismo, por mis cosas, por calmarme cuando el estrés que acarrea un trabajo de diploma me superaba. Gracias por todo, mi magia.

A mi familia:

Gracias por hacer por mi, todo lo que estuvo dentro de su alcance para que mi vida como estudiante no tuviera tropiezos, gracias especiales mamá, abuela, tío, papá.

Gracias a mi suegra, a la tía de mi novia, a mi cuñado, a la abuela de mi novia. Los considero también como mi familia porque así me han hecho sentir.

Y por supuesto, a mis amigos:

Que aunque generalmente se mencionan de último en los agradecimientos, y te dicen: "¡oye!, nos dejaste para el final". Son ellos las personas especiales que nunca te abandonan.

Ellos estuvieron a mi lado durante toda la carrera, dejando su esencia conmigo. Hicieron inolvidables los buenos momentos y mejoraron los malos. No puedo mencionar nombres porque no sería justo olvidarme de alguno. Y aunque, es cierto que los mencioné de último, tengan por seguro que lo mejor se reserva para el final, ya que los grandes finales quedan en la memoria de todos.

Dedicatoria

Dedico la presente investigación a la persona que incondicionalmente:
me dio consejos cuando no tenía respuestas,
secó mis lágrimas cuando estuve triste,
me impulsó cuando quedé sin fuerzas,
me corrigió cuando me equivoqué,
fue mi amiga cuando estuve solo,
y me dio la vida cuando no era nada.

Eres madre, padre y amiga, este resultado es tuyo,
gracias por todo mamá.

Resumen

Hoy en día, el desarrollo tecnológico ha contribuido a desarrollar productos informáticos capaces de controlar los procesos industriales, denominados sistemas de Supervisión, Control y Adquisición de Datos (SCADA) por sus siglas en Inglés. Estos sistemas requieren una elevada serie de configuraciones debido a sus dimensiones y complejidad. Actualmente el módulo Configuración del proyecto SCADA Galba del centro CEDIN de la UCI, administra los cambios que se realizan en la configuración del sistema sin generar un reporte auditable de los mismos, dando paso a que exista una vulnerabilidad de seguridad desde el punto de vista de que, no hay forma de saber, una vez que se ha configurado el mismo, quién realizó, o cuándo se realizó este cambio. Este trabajo tiene como objetivo desarrollar un mecanismo que posibilite generar registros de seguridad a partir de los cambios realizados en la configuración del sistema. Se utilizó para el desarrollo, el lenguaje de programación C++. Como gestor de base de datos, PostgreSQL, como herramientas Eclipse, PgAdmin III y Visual Paradigm; el desarrollo fue guiado por la metodología XP. La aplicación de las pruebas de aceptación permitió comprobar que la solución desarrollada cumple con las necesidades. Con la aplicación del mecanismo de generación de trazas desarrollado, se brindará a los administradores del SCADA Galba, una fuente de información auditable sobre los cambios ocurridos en la configuración del sistema.

Palabras clave: configuración, recurso, registro, scada, seguridad, traza.

Índice

Índice de tablas

Índice de figuras

Introducción

Durante el presente siglo, se ha desarrollado de forma exponencial la automatización de las industrias, debido a la complejidad de los procedimientos manuales y la necesidad de tener un control estricto sobre un grupo de procesos vitales dentro de las instituciones. En respuesta a esto se han desarrollado los sistemas de Control, Supervisión y Adquisición de Datos, más conocidos por sus siglas en inglés como SCADA (*Supervisory Control and Data Acquisition*).

El papel fundamental de los sistemas mencionados anteriormente, es la obtención de información proveniente de dispositivos vinculados al área operacional de una empresa o industria, estos datos son transmitidos a una estación que supervisa los procesos y administra la información recibida. Además, un SCADA está en condiciones de prevenir y alertar situaciones inesperadas que pudieran producirse dentro de la empresa o industria.

El Centro de Informática Industrial (CEDIN) perteneciente a la Facultad 5 de la Universidad de las Ciencias Informáticas, desarrolla productos y servicios informáticos de automatización industrial, con un alto valor agregado y que cumplan las necesidades y expectativas de los clientes, potenciando la formación especializada y la investigación (1). En la actualidad tiene contrato con la industria petrolera de Venezuela (PDVSA), para la cual desarrolla y despliega el sistema SCADA Guardián del ALBA (Galba). El Galba es un sistema altamente distribuido compuesto por varios módulos y diversos procesos que se ejecutan en toda la red (2).

El sistema anteriormente mencionado, al igual que todos los SCADA, es altamente configurable, es decir, requiere de una elevada serie de ajustes para su óptimo funcionamiento, ya sea de componentes propios del sistema o de usuarios que interactúan en él, según los permisos asociados a los mismos. La responsabilidad de llevar a cabo la tarea de configuración del sistema, queda soportada por el módulo Configuración, el cual es el encargado de mantener informados, a los diferentes módulos restantes, de los cambios ocurridos en el sistema en general y hacerles llegar los cambios a quienes se vean afectados en este proceso.

Seguidamente se expone la **situación problemática**:

Cada uno de los cambios, configuraciones o ajustes mencionados anteriormente causan un impacto importante en el funcionamiento del sistema, por lo que se necesita hacer persistente la información sobre quién, cuándo, dónde y por qué, se realizaron dichos cambios, para así poder auditarlos de alguna forma. Sin embargo:

1. El módulo no cuenta con esa posibilidad, lo cual crea una vulnerabilidad ya que el SCADA Galba está expuesto a ataques que, en ocasiones, podrían provenir desde dentro, es decir cualquier operador podría desajustar o desconfigurar el sistema quedando impune ya que no se podría demostrar de ninguna manera.
2. Por otra parte si en algún momento ocurre alguna falla no sería sencillo o rápido llegar al punto clave que causó el problema, lo que ocasionaría pérdidas de todo tipo de recursos.

Como resultado de lo expuesto anteriormente se formula el siguiente **problema científico**: ¿Cómo generar información auditable sobre los cambios ocurridos en la configuración del SCADA Galba?

A partir del problema científico formulado se define como **objeto de estudio**, los mecanismos informáticos para la generación de registros de información de seguridad.

Por lo que para darle solución al problema planteado se deriva el siguiente **objetivo general**: Desarrollar un mecanismo que permita la generación y el almacenamiento de trazas a partir de los cambios ocurridos en la configuración del SCADA Galba.

Todo lo anterior permite identificar como **campo de acción**, la generación de trazas de seguridad dentro del SCADA Galba.

Para dar cumplimiento al objetivo propuesto se definen las siguientes **tareas de investigación**:

1. Elaboración del marco teórico de la investigación a partir del estado del arte existente sobre el tema.
2. Definición y diseño de una solución que de cumplimiento al objetivo general.
3. Implementación del mecanismo diseñado.
4. Documentación del código fuente siguiendo los estándares definidos por la herramienta Doxygen.
5. Realización de pruebas al mecanismo.
6. Solución de no conformidades.

Desde el punto de vista metodológico se emplearon los siguientes **métodos científicos**:

Métodos teóricos:

- ✓ **Método analítico-sintético:** Se utilizó este método para el estudio de las teorías y documentos más relevantes sobre el uso e importancia de las trazas de seguridad, permitiendo así, extraer los elementos más importantes sobre los mismos.
- ✓ **Análisis histórico-lógico:** Se utilizó este método para realizar el estudio del estado del arte acerca del tema en cuestión, analizando los antecedentes y las tendencias actuales en cuanto a la generación de trazas de seguridad y basado en estos datos, complementar las características necesarias y deseables para la solución que se propone.
- ✓ **Modelación:** Utilizado para definir y representar gráficamente la solución que se plantea.

Métodos empíricos:

- ✓ **Consulta de la información en todo tipo de fuentes**: Se utilizó para la elaboración del marco teórico de la investigación.
- ✓ **Entrevistas:** Se utilizaron 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 del mecanismo.
- ✓ **Pruebas de validación de funcionamiento:** Se utilizaron para apreciar la operacionalidad de la propuesta.

Por tanto se puede enunciar la siguiente **idea a defender**: Si se aplica un mecanismo de generación de trazas en el módulo Configuración, se brindará a los administradores del SCADA Galba, una fuente de información auditable sobre los cambios realizados en la configuración del sistema.

El presente Trabajo de Diploma está estructurado de la siguiente manera:

Capítulo 1: “Fundamentación teórica”. En este capítulo se analiza en profundidad el objeto de estudio del problema a resolver, se hace referencia a las soluciones existentes de dicho problema, además de hacer un análisis, comparación y selección de la metodología de desarrollo de software y las herramientas y tecnologías a emplear.

Capítulo 2: “Características y diseño del sistema”. En este capítulo se describen las fases de Exploración, Planificación e Iteraciones, propias de la metodología que se emplea. Se obtienen las

historias de usuario y los requisitos no funcionales de la aplicación. También se diseña la aplicación mediante la selección de la arquitectura de software, la selección de patrones de desarrollo y la definición de las tarjetas CRC (Clase-Responsabilidad-Colaboración).

Capítulo 3: "Implementación y pruebas". En este capítulo se detallan los aspectos relacionados con la implementación de la solución propuesta, donde se incluyen las tareas de ingeniería, la explicación de las funcionalidades, los estilos de codificación y documentación. Además se comprueba el impacto que causa el mecanismo en el módulo Configuración, haciendo uso de pruebas de rendimiento y se valida la solución propuesta mediante las pruebas de aceptación.

Capítulo 1: “Fundamentación teórica”

1.1 Introducción

En el presente capítulo se abordan de manera general los aspectos fundamentales relacionados con la investigación. Se mencionan las soluciones existentes del problema de investigación. Se definen las principales características y conceptos asociados a la generación de trazas o registros de seguridad, así como su importancia dentro del módulo Configuración del sistema SCADA Guardián del ALBA. Además se mencionan definiciones generales sobre estos sistemas y también se hace referencia a la metodología, herramientas y tecnologías empleadas para dar solución al objetivo planteado.

1.2 Sistemas SCADA

SCADA viene de las siglas: “Supervisory Control And Data Acquisition”; es decir, hace referencia a un sistema de adquisición de datos con control supervisor. Tradicionalmente se define a un SCADA como un sistema que permite supervisar una planta o proceso por medio de una estación central que hace de master (llamada también estación maestra o unidad terminal maestra, MTU) y una o varias unidades remotas (generalmente RTUs) por medio de las cuales se hace control/adquisición de datos hacia/desde el campo (3).

Otra definición de sistema SCADA es: tecnología que posibilita a un usuario recolectar datos de uno o más medios distantes y enviar instrucciones de control limitadas a estos medios. Un SCADA hace que sea innecesario que un operador visite frecuentemente localizaciones remotas cuando estas operan normalmente. SCADA incluye una interfaz de operación y una de operación de datos (4).

Si bien las topologías sobre las que se sustentan los sistemas SCADA se han adecuado a los servicios de los sistemas operativos y protocolos actuales, las funciones de adquisición de datos y supervisión no han variado mucho respecto a las que proponían en sus inicios (3).
Esquemáticamente, un sistema SCADA conectado a un proceso automatizado consta de las siguientes partes (3):

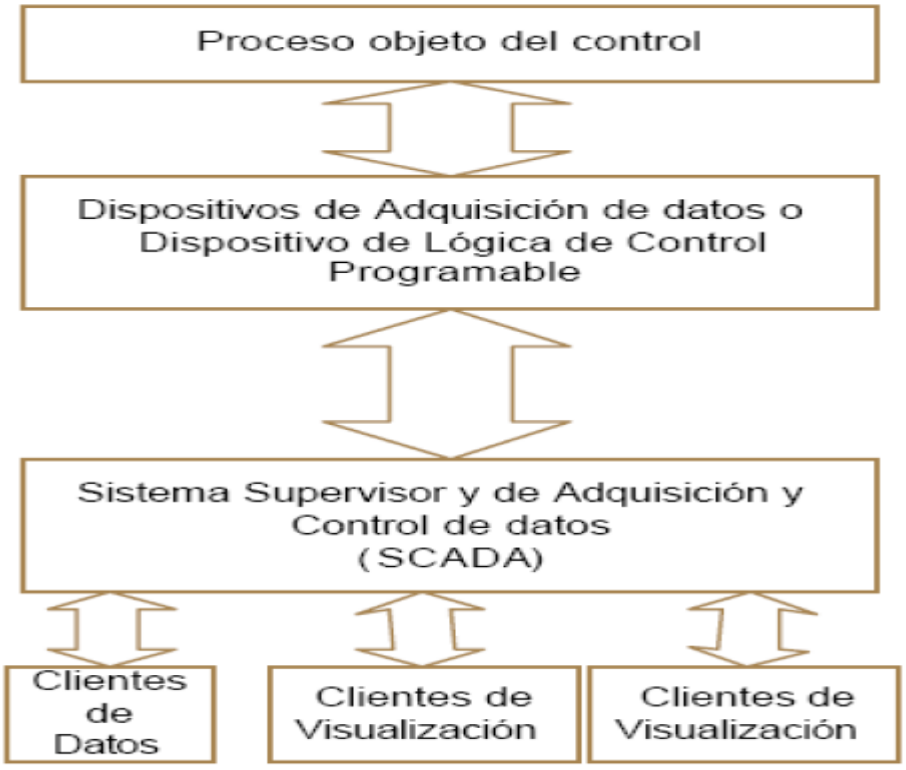


Figura 1: Esquema básico de un sistema SCADA (3).

- ✓ **Proceso objeto del control:** Es el proceso que se desea supervisar. En consecuencia, es el origen de los datos que se requieren colectar y distribuir.
- ✓ **Adquisición de datos:** Son un conjunto de instrumentos de medición dotados de alguna interfaz de comunicación que permita su interconexión.
- ✓ **SCADA:** Combinación de hardware y software que permita la colección, visualización y análisis de los datos proporcionados por los instrumentos.
- ✓ **Clientes:** Conjunto de aplicaciones que utilizan los datos obtenidos por el sistema SCADA.

1.2.1 Funciones principales de un sistema SCADA

Las funciones principales de un sistema SCADA son las que se describen a continuación (5):

- ✓ **Supervisión remota de instalaciones:** Mediante esta función, el usuario es capaz de conocer el estado de las instalaciones bajo su responsabilidad y coordinar eficientemente las labores de producción y mantenimiento en el campo. El intervalo de recolección

periódica de la información del campo depende de las dimensiones, pero generalmente está en el orden de unos cuantos milisegundos.

- ✓ **Control remoto de instalaciones:** Los sistemas SCADA permiten activar o desactivar equipos remotos (por ejemplo: interruptores, transformadores, bombas, válvulas, compresores, etc.) de manera automática o a solicitud del operador. Igualmente es posible realizar ajustes en parámetros en lazos de control analógicos (punto de consigna).
- ✓ **Procesamiento de información:** En algunos casos, los datos capturados requieren procesamiento adicional, a efectos de consolidar información proveniente de diferentes lugares remotos, como por ejemplo de balance de masa entre diferentes instalaciones.
- ✓ **Presentación de gráficos dinámicos:** Esto se refiere al despliegue de pantallas con el diagrama del proceso conteniendo información instantánea del comportamiento del mismo.
- ✓ **Generación de reportes:** Los sistemas SCADA permiten la generación automática o a petición, de reportes impresos de producción y balances.
- ✓ **Presentación de alarmas:** Mediante esta función se alerta al operador sobre la ocurrencia de condiciones anormales o eventos que pudieran requerir su intervención. Normalmente, la criticidad del evento o alarma se indica mediante el uso de colores y/o señales auditivas. Las alarmas se registran para análisis posteriores.
- ✓ **Almacenamiento de información histórica:** Los sistemas SCADA permiten registrar y almacenar información operacional y alarmas. Por ejemplo, se pueden llevar datos de los últimos 5 minutos, 1 hora, 1 día, 1 mes y hasta un año.
- ✓ **Presentación de gráficos de tendencias:** Con información en tiempo real o histórico, se pueden construir gráficos e inferir el comportamiento de variables operacionales en el tiempo.
- ✓ **Programación de eventos:** Se refiere a la posibilidad de programar en el tiempo la generación de reportes, despliegue de diagramas del proceso o activación de tareas o comandos del sistema.

1.3 SCADA Guardián del ALBA (Galba)

El GALBA es un sistema distribuido en módulos que trabajan de manera conjunta posibilitando el funcionamiento del mismo como un todo. Estos módulos se encuentran interconectados a través de un software para la distribución de los servicios en la red, conocido como "middleware" o

software de comunicación entre aplicaciones. La distribución de los módulos existentes en el SCADA permite obtener configuraciones escalables en dependencia de los requisitos que presente cada aplicación. Está dividido en varios subsistemas entre los que se encuentran (6):

- ✓ **Middleware:** Es la capa de software, que se encarga de la comunicación entre los diferentes módulos que forman parte del sistema. Este módulo tiene como finalidad proporcionar la capa de comunicación de alto nivel, tanto sincrónica, como asincrónica, para la comunicación de todos los módulos que conforman el sistema SCADA.
- ✓ **Adquisición:** Es el encargado de la adquisición, recepción, procesamiento y distribución de los datos provenientes del campo.
- ✓ **Configuración:** El servicio de configuración está formado por un grupo de componentes cada uno con una tarea específica y una base de datos que contendrá las configuraciones de cada uno de los módulos que conformen el proyecto activo. Es el encargado de almacenar, persistir y suministrar la información base para el funcionamiento de los demás módulos del SCADA.
- ✓ **Almacenamiento de datos históricos:** Es el encargado de almacenar la información del sistema para que posteriormente pueda ser empleada, por ejemplo, en generación de reportes, tendencias o en gestión de producción. La base de datos histórica (BDH) contendrá la información persistente de los datos recolectados de los dispositivos.
- ✓ **Seguridad:** Proporciona las funcionalidades necesarias para garantizar el trabajo autorizado a usuarios y módulos, además brinda las herramientas necesarias para la protección contra ataques maliciosos o involuntarios al sistema por parte de personas o recursos, tales como fallas de energía, problemas de red o servidores.
- ✓ **Visualización o HMI:** Se encarga de representar en un ordenador, los procesos que ocurren en el campo en tiempo real, muestra los componentes implicados, los sensores, las estaciones remotas, y el sistema de comunicación dándole al operador diferentes niveles de control en dependencia de sus niveles de privilegios. Este módulo permite al operador el contacto directo con el sistema y realizar la supervisión y el control del proceso en general.

1.3.1 Módulo Configuración del SCADA Galba

Entre las prestaciones del módulo Configuración se encuentra las siguientes (7):

- ✓ Guardar configuración en el servidor.
- ✓ Descargar configuración desde el servidor.
- ✓ Garantizar intercambio de información con otros módulos.
- ✓ Realizar configuración operacional.
- ✓ Realizar configuración en caliente.
- ✓ Gestionar configuraciones seguras.
- ✓ Deshacer los últimos cambios realizados.

- ✓ Para lograr un funcionamiento eficiente del mismo es necesario garantizar la escalabilidad del módulo, de manera que soporte desde una pequeña configuración hasta una configuración de gran complejidad y tamaño. El módulo debe ser lo más robusto posible para soportar cualquier problema de seguridad que atente contra la estabilidad del sistema. Además debe soportar la configuración en frío, en caliente y operacional. Estas configuraciones son explicadas a continuación (7):

- ✓ **Configuración en frío:** Cada uno de los módulos del sistema se inicia y solicitan sus parámetros de arranque al servidor de configuración y posteriormente se sincronizan. Todos los datos y archivos temporales son limpiados, excepto los valores almacenados en el servidor de datos históricos, que se mantienen de ejecuciones anteriores. Al finalizar esta acción el sistema se encuentra en la condición de inicializado. Para los módulos la configuración en frío implica que deben reiniciarse y volver a leer toda la configuración, lo cual es solventado con la configuración en caliente y operacional.

- ✓ **Configuración en caliente:** El sistema se encuentra en ejecución y se realizan modificaciones en la configuración desde una interfaz de edición. Toda configuración en caliente se deriva o está asociada a una configuración actual, de modo que el sistema se actualiza con los nuevos datos de configuración siguiendo los cambios realizados y no inicializándose como en el caso de la configuración en frío, este proceso ocurre sin afectar aquellos elementos del sistema que no estén involucrados en los cambios.

- ✓ **Configuración operacional:** El operador realiza cambios en el ambiente de ejecución del sistema (por lo general cambios mínimos en la configuración que no impliquen crear ni eliminar). El sistema se actualiza con los nuevos datos de configuración sin afectar la ejecución de este.

- ✓ Cada una de las configuraciones mencionadas anteriormente causa un impacto importante en el funcionamiento del sistema, por lo cual se hace necesario contar con un registro persistente de todos estos cambios. Es en este punto donde se hace necesario un mecanismo generador de trazas o registros de seguridad.

1.4 Traza

Según el diccionario de la real academia española la palabra **traza** significa, rastro de una persona o cosa.

Una traza es un registro oficial de eventos durante un rango de tiempo en particular. Para los profesionales en seguridad informática es usado para registrar datos o información sobre quién, qué, cuándo, dónde y por qué un evento ocurre para un dispositivo o aplicación en particular. La mayoría de las trazas son almacenadas o desplegadas en el formato estándar, el cual es un conjunto de caracteres para dispositivos comunes y aplicaciones. De esta forma cada traza generada por un dispositivo en particular puede ser leída y desplegada en otro diferente (2).

Teniendo en cuenta la investigación realizada, se puede afirmar lo siguiente:

Con frecuencia, se debe verificar que los programas, realizan exactamente las funciones previstas, y no otras. Para ello se apoya en productos software (algunos de los más conocidos se mencionarán en el acápite *1.5 "Soluciones existentes"*) que, entre otras funciones, rastrean los caminos que siguen los datos a través del programa, generando trazas, las cuales son utilizadas para comprobar la ejecución de las validaciones de datos previstas. Las mismas no deben modificar en absoluto el sistema.

Si la herramienta auditora produce incrementos apreciables de carga, se convendrá de antemano las fechas y horas más adecuadas para su empleo.

1.4.1 Formato estándar de representación de trazas

Mediante la investigación realizada se pudo arribar a la conclusión de que la forma de representación de trazas no es rígida, es decir se puede definir según las necesidades de la aplicación que se esté desarrollando. No obstante en este subtema se brinda una breve explicación sobre la representación formal.

Según propone "*The World Wide Web Consortium*" mejor conocido como W3C, el cual es un consorcio internacional que produce recomendaciones para la Web, la siguiente sintaxis muestra el formato estándar de representación de una traza.

origen id usuario fecha:hora petición código_estado bytes

***origen*:** Representa la dirección IP o nombre de la estación de trabajo del cliente que hizo la solicitud de recursos.
***id*:** Indica el elemento usado para identificar al cliente que realiza la solicitud.
***usuario*:** Indica el nombre de usuario utilizado por el cliente para la autenticación.
***fecha:hora*:** Indica la fecha y hora de la solicitud.
***petición*:** Indica la solicitud realizada por el cliente.
***código_estado*:** Indica el éxito o el fracaso de la petición.
***bytes*:** Contiene el número de bytes de datos transferidos como parte de la solicitud.
El campo ***fecha:hora*** posee también un formato estándar el cual es el siguiente:
[**dd/MMM/yyyy:hh:mm:ss +-hhmm**]
***dd*:** Representa el día del mes.
***MMM*:** Representa el mes.
***yyyy*:** Representa el año.
***:hh*:** Representa la hora.
***:mm*:** Representa el minuto.
***:ss*:** Representa el segundo.
***+-hhmm*:** Representa la zona horaria.

El siguiente ejemplo muestra estos campos poblados con valores para su mejor comprensión:
10.58.22.33 - jaoramas [23/Mar/2015:14:28:05 +0500] "GET /index.html HTTP/1.0" 200 1043
El símbolo "-" en algún campo indica que el dato no se obtuvo.

A continuación se muestra el formato que se utilizará en la solución propuesta. El mismo hará uso de algunos elementos del estándar mencionado, además se le agregarán otros elementos (resaltados en negrita) indispensables para resolver el problema en cuestión.

usuario**;** fecha_hora**; tipo_traza; tipo_recurso; id_recurso; campo; valor_anterior; valor_actual;**

tipo_traza: Indica el tipo de traza que se generará. Los tipos de traza son explicados en el acápite *2.4.2 "Arquitectura de software"*.

tipo_recurso: Especifica el tipo de recurso sobre el cual se ha generado la traza. En el acápite *2.4.2 "Arquitectura de software"* se explica el significado de recurso dentro del contexto de la investigación.

id_recurso: Identificador del recurso sobre el cual se ha generado la traza.

campo: Muestra el campo del recurso que ha sido modificado para el cual se genera la traza (se omite para las trazas de tipo *Adición* y *Eliminación*).

valor_anterior: Indica el valor anterior, del campo perteneciente al recurso que ha sido modificado, para el cual se genera la traza (se omite para las trazas de tipo *Adición* y *Eliminación*).

valor_actual: Indica el valor actual, del campo perteneciente al recurso que ha sido modificado, para el cual se genera la traza (se omite para las trazas con tipo *Adición* y *Eliminación*).

1.5 Soluciones existentes

Existen en el mundo aplicaciones informáticas denominadas sistemas de Información de Seguridad y Gestión de Eventos, más conocidas por sus siglas en inglés como SIEM (Security Information and Event Management). Este es un término para los productos de software y servicios que combinan la gestión de la información de seguridad y la gestión de eventos de seguridad. Estos sistemas poseen mecanismos de generación de trazas o registros de seguridad. Algunos de los más importantes se mencionarán a continuación.

1.5.1 LogRhythm

LogRhythm es una solución enfocada a la concentración de trazas, análisis de las mismas, correlación y monitoreo de eventos en tiempo real. Automatiza la recopilación, organización, análisis y archivo de todos los datos del registro. Esta herramienta automáticamente recoge, analiza y normaliza la información de trazas a través de la red y alerta a los departamentos de tecnologías de la información (TI) acerca de las posibles actividades sospechosas, así como de problemas que requieren atención inmediata. Para reducir los costos en cumplimiento, prevé la regulación específica de la presentación de informes y archivos. Cuenta con un fuerte registro de almacenamiento de datos, LogMart, que permite realizar investigaciones y presentar informes en línea a través de semanas, meses, o incluso un año completo de registro de trazas (8).

1.5.2 Splunk

Splunk es un software para buscar, monitorizar y analizar datos generados por máquinas de aplicaciones, sistemas e infraestructura de tecnologías de la información a través de una interfaz web. Splunk captura, indexa y correlaciona en tiempo real, almacenándolo todo en un repositorio donde busca para generar gráficos, alertas y paneles fácilmente definibles por el usuario. El objetivo de Splunk es hacer los datos accesibles a toda la organización, permitiendo la identificación de patrones, realización de medidas, diagnosis de problemas y provisión de inteligencia (Business Intelligence) a cualquier parte del negocio (9).

1.5.3 Monitor de servicios del SCADA Galba

El monitor de servicios del SCADA Guardián del ALBA, es una interfaz gráfica que permite supervisar y controlar los servicios de dicho sistema que se ejecutan en una PC, ya sea cliente o servidora. Además esta interfaz permite modificar los atributos de configuración necesarios para el buen funcionamiento de los módulos y realizar una visualización de los eventos (trazas) producidos por éstos.

Es muy útil tener una interfaz que pueda mostrar trazas y realizar ordenamiento, búsquedas especializadas y análisis avanzados que permitan detectar anomalías de funcionamiento de una manera ágil (2).

El monitor de servicios tiene una vista que permite visualizar las trazas generados por cada módulo, dichos trazas se adquieren desde los ficheros que se encuentran en las carpetas de trazas del Galba. Los eventos se muestran en una vista en forma de tabla que describe cada uno de ellos, donde se destaca la fecha y hora en que sucedieron, el texto que describe el evento y el tipo de evento. Además contiene funcionalidades que permiten limpiar o guardar las trazas del módulo que se está auditando así como uno para realizar análisis avanzado de los mismos. Para acceder a esta vista se debe navegar por el árbol y elegir el módulo al cual se desea auditar (2).

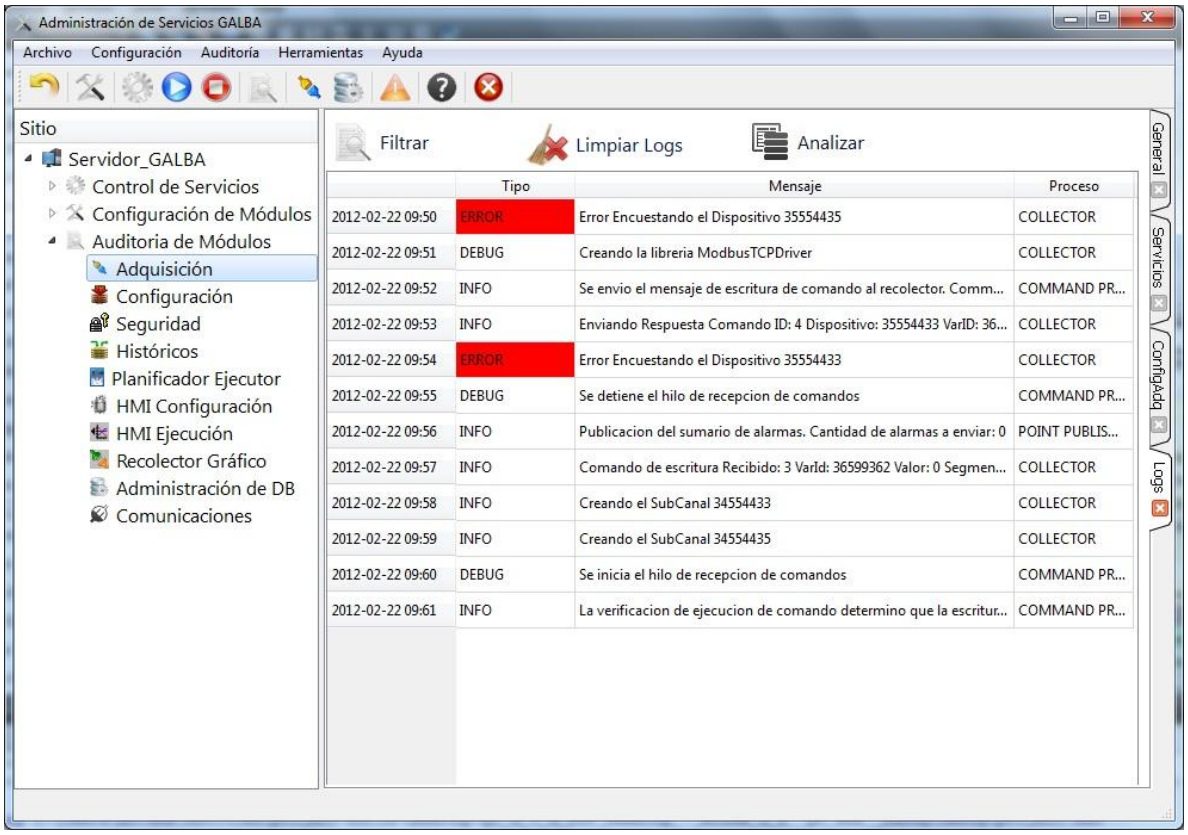


Figura 2: Interfaz para la auditoría de trazas (2).

Esta anterior característica hace destacable para la presente investigación, la aplicación mencionada, ya que podrá ser utilizada para realizar auditorías sobre la información brindada por el mecanismo de generación de trazas.

1.5.4 Análisis de las soluciones existentes

Luego de un análisis a los sistemas SIEM *LogRhythm* y *Splunk*, se concluye que, no pueden utilizarse para dar solución al problema, debido a que:

- ✓ Tienen un alto precio de adquisición, lo cual no los hace factible para un país sin un alto desarrollo económico como Cuba.
- ✓ Son sistemas privativos por lo cual no es posible estudiar su código fuente con el objetivo de analizarlo y desarrollar uno propio.

- ✓ Se debe mencionar además que son aplicaciones generalmente diseñadas para trabajar en entornos web, lo cual las hace incompatibles teniendo en cuenta que debemos acoplar el mecanismo a un sistema escritorio.

Además se analizó el *Monitor de servicios del Galba,* concluyendo que será la herramienta utilizada para auditar las trazas generadas por el mecanismo.

1.6 Metodología

En un proyecto de desarrollo de software la metodología define quién debe hacer qué, cuándo y cómo debe hacerlo. No existe una metodología de software universal. Las características de cada proyecto (equipo de desarrollo, recursos) exigen que el proceso sea configurable. Existen varias metodologías capaces de guiar este complejo proceso. Por una parte están las llamadas metodologías tradiciones que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Este tipo de metodologías son más eficaces y necesarias cuanto mayor es el proyecto que se pretende realizar respecto a tiempo y recursos que son necesarios emplear, donde una gran organización es requerida (10).

Por otra parte están las llamadas metodologías ágiles las cuales se encargan de valorar al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados, hacen mucho más importante crear un producto software que funcione, que escribir mucha documentación. Además es más importante la capacidad de respuesta ante un cambio. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad (10).

1.6.1 Metodología tradicional: RUP

Como parte del entorno de trabajo del centro CEDIN, el uso de la metodología tradicional Proceso Racional Unificado también conocido por sus siglas en Inglés como RUP (Rational Unified Process) está dentro de sus políticas de desarrollo. Esta metodología presenta tres características fundamentales que la definen (11):

- ✓ **Dirigido por casos de uso:** Un caso de uso es un fragmento de funcionalidad del sistema que proporcional al usuario un resultado importante. Los casos de uso representan los requisitos funcionales.
- ✓ **Centrado en la arquitectura:** Los arquitectos deben diseñar el sistema de forma tal que el sistema evolucione, no solo durante la etapa inicial sino también en las generaciones venideras.
- ✓ **Iterativo e incremental:** En los sistemas grandes es práctico dividir el trabajo en partes más pequeñas o mini -proyectos, donde cada uno es una iteración que posteriormente se convierte en un incremento o crecimiento del producto.

Sin embargo, comprende una gran cantidad de artefactos a generar, además posee gran cantidad de roles lo cual no lo hace efectivo para equipos de trabajo pequeños, y que necesite entregarse en poco tiempo (11).

Luego de haber realizado un análisis de esta metodología se pudo apreciar que su aplicación no sería efectiva en el desarrollo del mecanismo de generación de trazas ya que no es un sistema con un gran número de casos de usos, roles y requisitos funcionales. Para el desarrollo del mismo se plantea la utilización de una metodología ágil de desarrollo de software. A continuación se analizan las más utilizadas y reconocidas en el mundo.

1.6.2 Metodología ágil: SCRUM

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

El marco de trabajo SCRUM consiste en los Equipos SCRUM, roles, eventos, artefactos y reglas asociadas. Cada componente dentro del marco de trabajo sirve a un propósito específico y es esencial para el éxito de SCRUM y para su uso. Las reglas de SCRUM relacionan los eventos, roles y artefactos, gobernando las relaciones e interacciones entre ellos (10).

1.6.3 Metodología ágil: Programación Extrema (XP)

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios (12).

El ciclo de vida de un proyecto XP incluye, al igual que las otras metodologías,
entender lo que el cliente necesita, estimar el esfuerzo, crear la solución y entregar el producto final al cliente. Sin embargo, XP propone un ciclo de vida dinámico, donde se admite expresamente que, en muchos casos, los clientes no son capaces de especificar sus requerimientos al comienzo de un proyecto. Por esto, se trata de realizar ciclos de desarrollo cortos (llamados iteraciones), con entregables funcionales al finalizar cada ciclo. En cada iteración se realiza un ciclo completo de análisis, diseño, desarrollo y pruebas, pero utilizando un conjunto de reglas y prácticas que caracterizan a XP (13).
Típicamente un proyecto con XP lleva 10 a 15 ciclos o iteraciones. La siguiente figura esquematiza los ciclos de desarrollo en cascada e iterativos tradicionales (por ejemplo, incremental o espiral), comparados con el de XP (13).

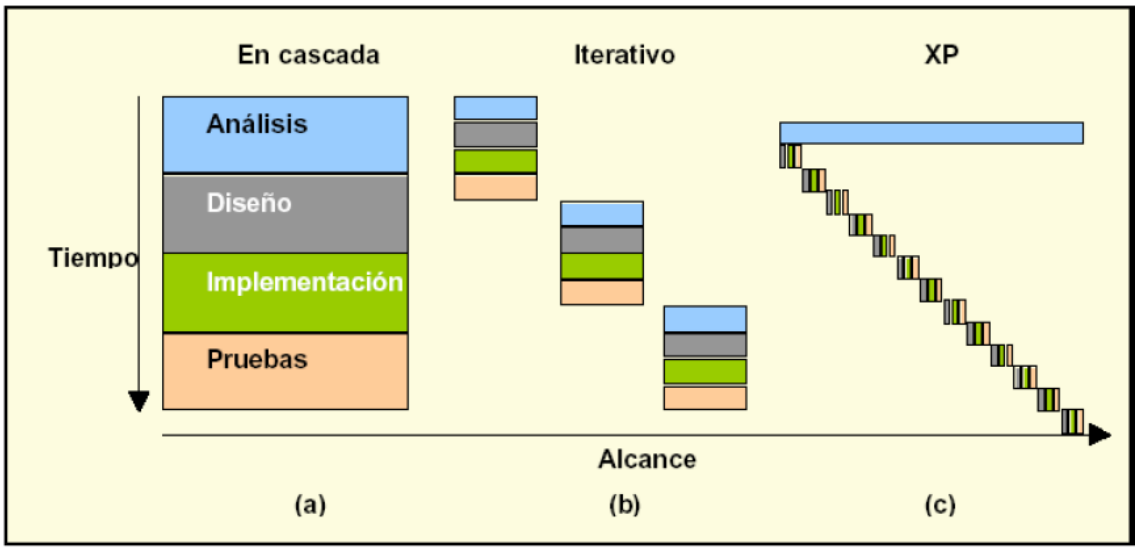


Figura 3: Ciclos de desarrollo (14).

Si bien el ciclo de vida de un proyecto XP es muy dinámico, se puede separar en fases (10):

1. Exploración.
2. Planificación de la Entrega.
3. Iteraciones.
4. Producción.
5. Mantenimiento.
6. Muerte del Proyecto.

1.6.4 Selección de la metodología a emplear

Luego de analizar las características de las metodologías mencionadas anteriormente, se selecciona XP para que guíe el proceso de desarrollo, debido a que:

- ✓ Se necesita desarrollar un sistema pequeño, el cual no tendrá documentación excesiva y estará centrado fundamentalmente en la implementación.
- ✓ Es la metodología que mejor se ajusta a las necesidades del proyecto en cuanto a recursos técnicos, humanos y tiempo de desarrollo.
- ✓ El equipo de trabajo es pequeño, el cliente forma parte de él y está en constante interacción con la aplicación.

- ✓ Permite que en cualquier momento del proceso de desarrollo de la aplicación, se pueda regresar a operaciones anteriores sin afectar el ciclo de vida del software.

1.7 Herramientas y tecnologías

La calidad de un producto de software está determinada en gran medida por la calidad del proceso utilizado para desarrollarlo y mantenerlo. A continuación se enunciarán las herramientas y tecnologías utilizadas. La selección de las mismas, se basa fundamentalmente en herencias del proyecto y las políticas de desarrollo del módulo para el cual se desarrolla el mecanismo.

1.7.1 Lenguaje de modelado UML

UML (Unified Modeling Language) es desde finales de 1997 un lenguaje de modelado visual que se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema de software (15).

UML constituye un lenguaje más expresivo, claro y uniforme que los anteriores definidos para el diseño orientado a objetos, no brinda el total éxito de los proyectos pero si mejora sustancialmente el desarrollo de los mismos al posibilitar una nueva y fuerte integración entre las herramientas, los procesos y los dominios. Además UML posee una corrección fiable de errores en todas las etapas de la construcción del software y es aplicable para tratar asuntos en sistemas complejos de misión crítica, tiempo real y cliente-servidor (16).

1.7.2 Lenguaje de programación C++

Un lenguaje de programación es utilizado para controlar el comportamiento físico y lógico de una máquina.

C++ es un lenguaje de programación de utilidad general diseñado para que el programador serio haga su trabajo de modo más agradable. Salvo algunos detalles menores C++ es un súper conjunto del lenguaje de programación C. Este lenguaje conserva la capacidad de C para manejar en forma eficiente los objetos fundamentales de la máquina (bits, bytes, palabras, direcciones, etc.). Esto hace posible una implantación con un alto grado de eficiencia de los tipos definidos por el usuario (17).

En la presente investigación se hará uso especialmente de los contenedores de la biblioteca estándar del lenguaje.

1.7.3 Sistema Gestor de Base de Datos PostgreSQL

Los Sistemas Gestores de Bases de Datos (SGBD) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan (18).

PostgreSQL es un sistema gestor de bases de datos distribuido bajo la licencia BSD y con su código fuente disponible libremente. Posee características avanzadas como consultas complejas, llaves foráneas, vistas, integridad transaccional, control de concurrencia multiversión y disparadores. Se ejecuta en casi todos los principales sistemas operativos: Linux, Unix, Mac OS y Windows. Se desempeña muy bien con grandes volúmenes de datos y con gran concurrencia en el sistema (19).

Proporciona gran variedad de tipos de datos. Además de los usuales: numérico, cadena y tipos de fecha, también soporta los tipos: booleanos y geométricos, tales como punto, segmento de línea, caja, polígono y círculo. Puede ser extendido por el usuario agregando nuevos tipos de datos, funciones y operadores. Está basado en la arquitectura cliente-servidor. Con más de dos décadas de desarrollo, PostgreSQL es uno de los servidores de base de datos más avanzados del mundo (20).

1.7.4 Entorno de Desarrollo Integrado Eclipse

Un Entorno de Desarrollo Integrado o IDE por sus siglas en inglés, es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código, cuyo objetivo es brindar un entorno para programar, en uno o varios lenguajes de programación, mediante un grupo de herramientas que así lo permiten (21).

Eclipse es un IDE de código abierto y multiplataforma. Mayoritariamente se utiliza para desarrollar lo que se conoce como "Aplicaciones de Cliente-Enriquecido", opuesto a las "Aplicaciones de Cliente-Liviano" basadas en navegadores. Es una potente y completa plataforma de programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java (22).

1.7.5 Herramienta CASE Visual Paradigm

Visual Paradigm es una herramienta CASE profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

Este software ayuda a una más rápida construcción de aplicaciones de calidad y a un menor coste (23).

Las características principales de esta herramienta son las que a continuación se enuncian (23):

- ✓ Soporte de UML.
- ✓ Ingeniería inversa: código a modelo, código a diagrama.
- ✓ Generación de código: modelo a código, diagrama a código.
- ✓ Generación de bases de datos: transformación de diagramas de entidad-relación en tablas de base de datos.
- ✓ Ingeniería inversa de bases de datos: desde Sistemas Gestores de Bases de Datos existentes a diagramas de entidad-relación.
- ✓ Editor de figuras.

1.7.6 Biblioteca Log4cxx

Log4cxx es la adaptación para el lenguaje C++ de Log4j, la cual es una biblioteca de código abierto desarrollada en Java por la Fundación de Software Apache (Apache Software Foundation) que permite a los desarrolladores de software escribir mensajes de registro, cuyo propósito es dejar constancia de una determinada transacción en tiempo de ejecución (24).

Log4cxx es un marco de trabajo rápido y flexible que permite filtrar los mensajes en función de su importancia. Está diseñado para que estos mensajes de registro puedan ser generados sin incurrir en un alto costo de rendimiento. El comportamiento de los registros puede ser controlado mediante la edición de un archivo externo de configuración, sin modificar el código de la aplicación (24).

Esta biblioteca será empleada para registrar en ficheros de texto, la información referente a los cambios ocurridos en la configuración del Galba.

Consideraciones parciales

En este capítulo se fundamentó teóricamente el fin de la investigación, se definieron las herramientas y tecnologías a utilizar, como metodología de desarrollo de software se eligió XP y para realizar el modelado del sistema la herramienta CASE Visual Paradigm, apoyándose en el Lenguaje Unificado de Modelado (UML). Se seleccionó Eclipse como IDE, C++ como lenguaje de programación; finalmente se escogió el gestor de base de datos PostgreSQL para el tratamiento

de los datos. De esta forma quedan las bases sentadas para comenzar con el diseño de la solución.

Capítulo 2: "Características y diseño del sistema"

2.1 Introducción

El presente capítulo abordará sobre el diseño del mecanismo para la generación de trazas a partir de los cambios ocurridos en la configuración del SCADA Galba, además de comprender el contexto e identificar las condiciones o capacidades que debe cumplir. También las fases de Exploración, Planificación e Iteraciones, todas estas etapas propias de la metodología de desarrollo utilizada. Se describen los artefactos obtenidos según la metodología. Además se especifican los requisitos funcionales y no funcionales que deberá cumplir la solución.

2.2 Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología (10).

2.2.1 Historias de usuario

Las historias de usuario (HU) son la técnica utilizada para representar y especificar los requisitos del software. Se trata de un conjunto de tablas en las cuales el cliente describe brevemente las características que el sistema debe poseer. Su tamaño no excede de unas pocas líneas de texto. El tratamiento de las HU es muy dinámico y flexible. Tienen el mismo propósito que los casos de uso, las describen los propios clientes tal y como ven ellos las necesidades del sistema. Las historias de usuario son descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración (10).

En el presente trabajo de diploma se definieron un total de 8 historias de usuario, cada una de ellas respondiendo a las diferentes funcionalidades solicitadas por el cliente, las cuales se realizarán en 3 iteraciones. Se clasificaron de la siguiente manera: 4 HU de prioridad alta, 2 HU de prioridad media y 2 HU de prioridad baja.

Además se seleccionó como unidad de medida para los puntos estimados, una semana.

Las HU se representan mediante tablas las cuales contienen las siguientes secciones:

- ✓ **Número:** Un número consecutivo, este permite identificar la HU.
- ✓ **Nombre:** Nombre que identifica la HU.
- ✓ **Referencia:** Es el conjunto de números de las diferentes HU de las cuales depende la que actualmente se encuentra en desarrollo.
- ✓ **Prioridad:** Esta característica es dada por el cliente con los valores: alta, media o baja en dependencia de la importancia y orden en que desean que sean implementadas.
- ✓ **Riesgo en desarrollo:** Esta característica es dada por el programador con los valores: alto, medio o bajo en dependencia de la complejidad que represente la implementación de la HU.
- ✓ **Iteración asignada:** Número de la iteración en la cual se desarrollará la HU.
- ✓ **Puntos estimados:** Tiempo estimado en semanas que se le asignará.
- ✓ **Descripción:** breve descripción del proceso que define la historia.

En las tablas 1 a la 8 se muestran las historias de usuario.

Tabla 1: Historia de usuario No. 1.

Historia de usuario	
Número: 1	**Referencia:** -
Nombre: Generar trazas de tipo adición.	
Prioridad: Alta	**Iteración asignada:** 1
Riesgo en desarrollo: Alto	**Puntos estimados:** 1
Descripción: El mecanismo debe permitir generar trazas de seguridad de tipo *adición* una vez que el operador cargue una configuración en el HMI Editor y en la misma se hallan agregado nuevos recursos.	

Tabla 2: Historia de usuario No. 2.

Historia de usuario	
Número: 2	**Referencia:** -
Nombre: Generar trazas de tipo modificación.	
Prioridad en negocio: Alta	**Iteración asignada:** 1

Riesgo en desarrollo: Alto	**Puntos estimados:** 1
Descripción: El mecanismo debe permitir generar trazas de seguridad de tipo *modificación* una vez que el operador cargue una nueva configuración en el HMI Editor y en la misma se halla hecho algún cambio sobre algún tipo de recurso ya existente.	

Tabla 3: Historia de usuario No. 3.

Historia de usuario	
Número: 3	**Referencia:** -
Nombre: Generar trazas de tipo eliminación.	
Prioridad: Alta	**Iteración asignada:** 1
Riesgo en desarrollo: Alto	**Puntos estimados:** 1
Descripción: El mecanismo debe permitir generar trazas de seguridad de tipo *eliminación* una vez que el operador cargue una nueva configuración en el HMI Editor y en la misma se halla eliminado algún recurso existente.	

Tabla 4: Historia de usuario No. 4.

Historia de usuario	
Número: 4	**Referencia:** 1, 2, 3
Nombre: Detectar cambios en la configuración del SCADA Galba.	
Prioridad: Alta	**Iteración asignada:** 1
Riesgo en desarrollo: Alto	**Puntos estimados:** 3
Descripción: El mecanismo debe detectar si han ocurrido cambios en una configuración nueva cargada al SCADA, con respecto a una configuración previamente existente.	

Tabla 5: Historia de usuario No. 5.

Historia de usuario	
Número: 5	**Referencia:** 1, 2, 3, 4
Nombre: Enviar trazas hacia el módulo de histórico.	
Prioridad: Media	**Iteración asignada:** 2
Riesgo en desarrollo: Medio	**Puntos estimados:** 1

Descripción: El mecanismo debe permitir, que una vez generadas las trazas, sean enviadas hacia el módulo de histórico para hacer persistente esta información.

Tabla 6: Historia de usuario No. 6.

Historia de usuario	
Número: 6	**Referencia:** 1, 2, 3, 4
Nombre: Analizar la descripción de una traza.	
Prioridad: Media	**Iteración asignada:** 2
Riesgo en desarrollo: Medio	**Puntos estimados:** 0.6
Descripción: Una vez generadas las trazas, son enviadas hacia el módulo de histórico como cadenas de texto. Por este motivo se necesita analizar la descripción de las mismas para poder desglosar esta información y luego almacenarla en la base de datos.	

Tabla 7: Historia de usuario No. 7.

Historia de usuario	
Número: 7	**Referencia:** 1, 2, 3, 4
Nombre: Imprimir trazas en consola.	
Prioridad: Baja	**Iteración asignada:** 3
Riesgo en desarrollo: Bajo	**Puntos estimados:** 0.2
Descripción: El sistema debe permitir, que una vez generadas las trazas, puedan imprimirse en la consola.	

Tabla 8: Historia de usuario No. 8.

Historia de usuario	
Número: 8	**Referencia:** 1, 2, 3, 4
Nombre: Almacenar trazas en fichero.	
Prioridad: Baja	**Iteración asignada:** 3
Riesgo en desarrollo: Bajo	**Puntos estimados:** 0.3
Descripción: El sistema debe permitir que una vez generadas las trazas, puedan ser almacenadas en un fichero de texto.	

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son las cualidades que debe tener el producto. Estos requisitos hacen que el producto sea atractivo, útil, rápido, fiable y seguro. Estas propiedades no son requeridas porque no son actividades funcionales del producto, pero son deseables, ya que el cliente espera que las actividades sean ejecutadas de cierta manera y con un específico grado de calidad. Los requisitos no funcionales no alteran la funcionalidad (25).

A continuación se muestran los requerimientos:

- ✓ **Usabilidad**

  RnF1: El módulo debe mantener el mismo procedimiento de arranque y puesta en marcha que poseía antes de agregar una nueva funcionalidad.

- ✓ **Confiabilidad**

  RnF2: El mecanismo debe garantizar que cuando ocurra un fallo en el sistema no se modifique la configuración anterior o no se guarde ninguna.

- ✓ **Eficiencia**

  RnF3: Uso mínimo de los recursos: El sistema debe hacer uso de la menor cantidad de recursos. Este trabaja con una gran cantidad de información, por lo que debe hacer uso eficiente de la memoria.

- ✓ **Soporte**

  RnF4: La programación del sistema debe estar orientada a objeto.

- ✓ **Restricciones de diseño**

  RnF5: El mecanismo debe ser de código abierto: El sistema debe ejecutarse en diversas plataformas de hardware y software independientemente del número y composición de los bits de la arquitectura.

  RnF6: Empleo del mismo conjunto de herramientas en el desarrollo del sistema: Se debe emplear el mismo conjunto de herramientas para el desarrollo del sistema, así como para futuros desarrollos. De manera tal que garantice la escalabilidad y compatibilidad entre los diferentes desarrollos.

2.3 Planificación

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman

acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente (10).

2.3.1 Velocidad del proyecto

Las estimaciones de esfuerzo asociado a la implementación de las historias las establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos (10).

A continuación se presenta la tabla 9 donde se resume la estimación del esfuerzo realizada por parte de los desarrolladores.

Tabla 9: Estimación del esfuerzo.

Historia de usuario	Puntos de estimación (semanas)
Generar trazas de tipo adición.	1.0
Generar trazas de tipo modificación.	1.0
Generar trazas de tipo eliminación.	1.0
Detectar cambios en la configuración del SCADA Galba.	3.0
Enviar trazas hacia el módulo de histórico.	1.0
Analizar la descripción de una traza.	1.0
Imprimir trazas en consola.	0.3
Almacenar trazas en fichero.	0.6

Después del análisis anterior se arribó a que el desarrollo total estimado es de 8.9 semanas lo que representa un total de 9 semanas aproximadamente.

2.3.2 Plan de iteraciones

Luego de identificar las HU y la estimación del esfuerzo por cada una de ellas, se procede a realizar el plan de iteraciones, en el cual estarán contenidas las HU en el orden a realizar por cada iteración según su orden de relevancia, así como la descripción y el total de semanas que durarán cada una de estas. A continuación se muestra el plan de iteraciones en la tabla 10:

Tabla 10: Plan de iteraciones.

Iteración	Descripción	Historias de usuario	Puntos de estimación (semanas)
1	La primera iteración tiene como objetivo llevar a cabo la implementación de las historias de usuario de prioridad alta, las cuales poseen funcionalidades que son indispensables para cubrir las necesidades del cliente y que inciden críticamente en el funcionamiento del mecanismo.	1, 2, 3, 4	7.0
2	En la segunda iteración se llevará a cabo la implementación de las historias de usuario de prioridad media, las cuales no son menos importantes para el sistema que las primeras, pero se decidió separarlas con el objetivo de organizar mejor el trabajo.	5, 6	2.0
3	Esta iteración tiene como objetivo implementar las HU de menor prioridad.	7, 8	1.0

2.3.3 Plan de entrega

A partir del plan de iteraciones descritas anteriormente se procede a realizar el plan de entrega, el cual tiene como objetivo dar a conocer las fechas de culminación de las iteraciones y sus correspondientes historias de usuario.

Tabla 11: Plan de entrega.

	Iteración # 1	Iteración # 2	Iteración # 3
Cantidad de HU	4	2	2
Fecha de entrega	20/04/2015	01/05/2015	08/05/2015

2.4 Diseño del sistema

La metodología XP no requiere de la representación del sistema mediante diagramas de clases utilizando UML para diseñar las aplicaciones, en este caso utiliza otras técnicas como las tarjetas CRC (Clase-Responsabilidad-Colaboración). Esto no implica que no se utilicen los diagramas para obtener una mejor visión y comunicación entre el equipo de trabajo, siempre y cuando su complejidad no sea alta y defina información importante.

2.4.1 Tarjetas Clase-Responsabilidad-Colaboración (CRC)

Las tarjetas Clase-Responsabilidad-Colaboración (CRC) permiten desprenderse del método basado en procedimientos y trabajar con una metodología basada en objetos, así el programador se centra y comienza a apreciar el desarrollo orientado a objetos (26).

Las tarjetas CRC representan objetos y se describen a partir de los siguientes elementos.

- ✓ **Clase**: Nombre de la clase a la cual pertenece la tarjeta.
- ✓ **Responsabilidades**: Describe cuales son las funcionalidad que deben ser implementadas por la clase.
- ✓ **Colaboración**: Enumera las diferentes clases con las cuales tiene relación la clase a la cual pertenece la tarjeta CRC.

A continuación se muestran las tarjetas CRC identificadas por el autor.

Tabla 12: Tarjeta CRC DataLog.h.

Tarjeta CRC	
Clase: DataLog.h	
Responsabilidades:	**Colaboración:**

Almacenar la información referente a una traza.	DataTypeDefinition.h

Tabla 13: Tarjeta CRC LogGenerator.h.

Tarjeta CRC	
Clase: LogGenerator.h	
Responsabilidades:	**Colaboración:**
Generar trazas de los puntos analógicos.	DataLog.h ConfigurationServer.h ConfigurationServerLayerImpl.h DataTypeDefinition.h DBConnection.h
Generar trazas de los puntos analógicos calculados.	
Generar trazas de los puntos analógicos de escritura.	
Generar trazas de los puntos digitales.	
Generar trazas de los puntos digitales calculados.	
Generar trazas de los puntos digitales de escritura.	
Generar trazas de los puntos tipo cadena.	
Generar las trazas de las alarmas de falla de comunicación.	
Generar las trazas de las alarmas de falla de ejecución de comando.	
Generar las trazas de las alarmas de falla de instrumento.	
Generar las trazas de las alarmas de nivel.	
Generar las trazas de las alarmas de no variación en el tiempo.	
Generar las trazas de las alarmas de cambio de estado no comandado.	
Generar las trazas de las alarmas de	

estado.	
Generar las trazas de los canales.	
Generar las trazas de los subcanales.	
Generar las trazas de los dispositivos.	
Generar las trazas de los parámetros de los subcanales.	
Generar las trazas de los parámetros de los dispositivos.	
Generar las trazas de las estructuras	
Generar las trazas de las estructuras genéricas.	
Generar las trazas de las condiciones de las alarmas.	
Generar las trazas de las condiciones aperiódicas.	
Generar las trazas de las condiciones periódicas.	
Generar las trazas de las condiciones de los puntos.	
Generar las trazas de las tareas.	
Generar las trazas de los módulos.	
Generar las trazas del proyecto.	
Generar las trazas de los gth.	
Generar las trazas de las conexiones.	
Generar las trazas de los menús.	
Generar las trazas de los eventos.	
Generar las trazas de los reportes.	
Generar las trazas de los despliegues.	
Generar las trazas de las relaciones HMI-Reporte.	

Generar las trazas de las relaciones HMI-Despliegue.	
Generar las trazas de los GOP.	
Generar las trazas de los perfiles.	
Generar las trazas de los usuarios.	
Generar las trazas de la relación perfil-GOP.	
Generar las trazas de la relación perfil-usuario.	
Generar las trazas de los recursos externos.	
Generar las trazas de los recursos nuevos.	
Generar las trazas de los recursos modificados.	
Generar las trazas de los recursos eliminados.	
Convertir enumerativos de recursos en cadenas.	

Tabla 14: Tarjeta CRC ParserLogDescription.h.

Tarjeta CRC	
Clase: ParserLogDescription.h	
Responsabilidades:	**Colaboración:**
Analizar la información de la descripción de una traza.	
Devolver el tipo de acción que define a la traza.	
Devolver el tipo de recurso para el cual se genera la traza.	
Devolver el identificador de la traza.	

Devolver el campo que fue modificado.	
Devolver el valor anterior del campo que se modificó.	
Devolver el valor actual del campo que se modificó.	

2.4.2 Arquitectura de software

Según el Instituto de Ingeniería Eléctrica y Electrónica (IEEE), la **arquitectura del software** es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución.

Arquitectura del módulo Configuración:

El módulo Configuración presenta la arquitectura cliente-servidor, donde los agentes de configuración actúan como clientes, solicitando permiso para leer o escribir en el servidor de configuración quien atiende las peticiones y otorga los permisos requeridos para la realización de las mismas. Se entiende por agente de configuración, a la biblioteca que le brinda el módulo Configuración al resto de los módulos del SCADA Galba, para que puedan obtener o modificar la configuración. Existe una biblioteca específica por cada tipo de módulo definido en el sistema. La labor fundamental de los agentes es abstraer a los módulos de la complejidad del proceso que es necesario realizar cuando se desea obtener o modificar la configuración. Seguidamente para una mejor comprensión de esta arquitectura se explica el proceso de configuración en caliente, uno de los más importantes (7):

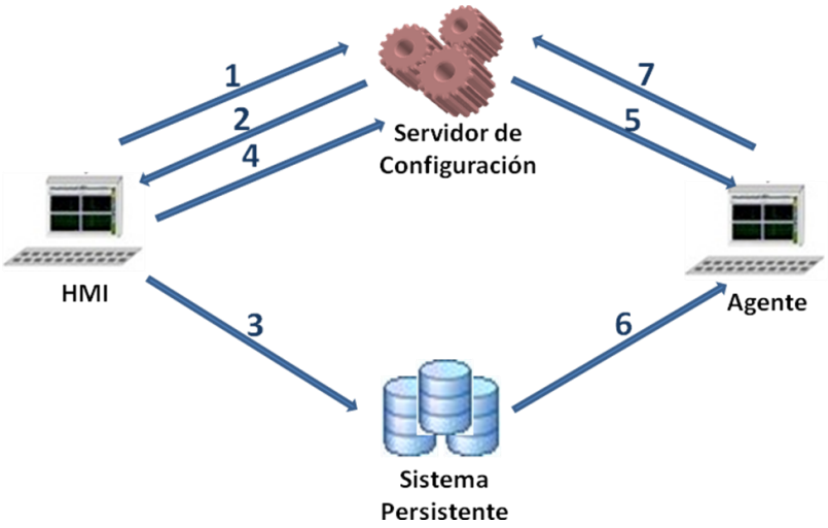


Figura 4: Proceso de configuración en caliente. SCADA Galba (7).

1. El agente correspondiente al HMI Edición solicita al servidor de configuración permiso para almacenar la nueva configuración en el formato persistente.
2. Cuando sea posible almacenar la nueva configuración (no existe ningún otro agente obteniendo información del formato persistente o modificando la misma) el servidor de configuración procede a darle permiso al agente para que se encargue de almacenar la nueva configuración.
3. El agente almacena la configuración en el formato persistente.
4. El agente le comunica al servidor de configuración que culminó la operación.
5. El servidor de configuración, analiza los cambios que fueron realizados y determina qué módulos deben ser informados de la ocurrencia de modificaciones en la configuración. A los módulos cuya configuración haya sido modificada por los cambios realizados, el servidor de configuración les notifica mediante el agente alertándolos de ello.
6. Los agentes de cada módulo cuya configuración fue modificada acceden a la información almacenada en el formato persistente y extraen los datos necesarios para la actualización del módulo.
7. Los agentes informan al servidor de configuración la culminación de la operación de obtención de la nueva configuración.

Dentro del proceso anterior se encuentra el mecanismo de generación de trazas, el cual se acoplará a la arquitectura del módulo, haciendo uso de la misma. El flujo de tareas ocurrirá de la siguiente manera:

1. Una vez que se carga una nueva configuración al SCADA Galba se procede a detectar cambios en dicha configuración (**HU4**), comparando la nueva configuración con la configuración existente hasta el momento, esta última es almacenada temporalmente en el sistema persistente. Para ello se analizan los cambios que han ocurrido con respecto a cada tipo de recurso[1], de estos existen 43 tipos y se realizan tres comprobaciones fundamentales para cada uno. Se verifica si se añadió, si se modificó, o si se eliminó alguno de ellos.
2. En dependencia de los cambios detectados con anterioridad se generan trazas de tipo *adición, modificación* o *eliminación* (**HU1, HU2, HU3**). Este proceso se basa en crear objetos de la clase *DataLog.h,* la cual modela la información referente a una traza, para luego almacenarlos en el contenedor mLogs. Un objeto de este tipo podrá ser construido de dos formas. La primera modelará las trazas de tipo adición y eliminación, contendrá el identificador del recurso y una cadena de texto con la información referente a la traza siguiendo el formato: ***Adición/Eliminación; Tipo:*** *nombre_recurso****; ID:*** *id_recurso****;***.
3. La segunda modelará las trazas de tipo modificación, la cual contendrá el identificador del recurso, el valor anterior al cambio, el valor actual y una cadena de texto con la descripción de la traza siguiendo el formato:

   Modificación; Tipo: *nombre_recurso****; ID:*** *id_recurso****; Campo:*** *campo_modificado****;***.
4. Luego de tener el contenedor mLogs con las trazas, estas se pueden imprimir en la consola (**HU7**) o almacenar en el disco duro en un fichero (**HU8**). Estas acciones estarán reguladas por el valor que tome el parámetro *tratamiento_logs* del script *configuracion.conf* el cual contiene la información necesaria que debe leer el servidor de configuración para iniciarse.
5. Además de las acciones anteriores, se deben enviar las trazas hacia el módulo de histórico (**HU5**), lo cual se hará en todo momento. Para ello se hace uso de la función *sendHistoricalLog* de la clase *ConfigurationServerLayerImpl.h* la cual es la encargada de

[1] Es la manera en que el módulo Configuración nombra a los elementos que maneja el SCADA, entiéndase: puntos, alarmas, canales, dispositivos, entre otros.

implementar las interfaces necesarias para enviar y recibir información en el servidor haciendo uso de la capa de comunicación.

2.4.3 Patrones de diseño

"En la terminología de objetos, el patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas. Muchos patrones ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problemas. (…) Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería de software. Todo lo contrario, intentan codificar el conocimiento, las expresiones y los principios ya existentes, cuanto más trillados y generalizados, mejor (27)."

Patrones GRASP

Los Patrones de Software para la Asignación General de Responsabilidades GRASP (General Responsabilty Assignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones. En la implementación de la aplicación se utilizaron los siguientes:

Alta cohesión: Se encarga de elevar el nivel en el que una clase tiene comportamientos concretos y relacionados con el fin para el cual fue creada. Una clase con baja cohesión hace muchas cosas no afines o realiza un trabajo excesivo. Estas a menudo representan un alto grado de abstracción o han asumido responsabilidades que deberían haber delegado a otros objetos.
Este patrón fue utilizado en el diseño de la aplicación de manera general, siguiendo la premisa de que cada clase debe contener operaciones que resuelvan necesidades a fines con ellas.

Experto: El propósito del patrón Experto es asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. La responsabilidad de la creación de un objeto o la implementación de un método debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento) (27).

Dicho patrón se evidencia en la definición de las clases de acuerdo a las funcionalidades que deben realizar a partir de la información que manejan.

Creador: Se encarga de identificar quien debe tener la responsabilidad de crear o instanciar objetos. Este patrón da soporte al bajo acoplamiento. En el presente caso, la clase LogGenerator será la encargada de crear objetos de tipo DataLog, a su vez tratará con los mismos y los contendrá.

Patrones GoF

Los patrones Banda de los Cuatro o GOF (Gang of Four) por sus siglas en inglés, indican soluciones técnicas basadas en Programación Orientada a Objetos (POO). En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje.

Puente/Bridge: Separar una abstracción de su implementación para permitir que ambos puedan variar independientemente. Se hace uso de este patrón en la aplicación, al re-implementar el método sendHistoricalLog de la clase ConfigurationServer.h.

Solitario/Singleton: Garantizar que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma. Se hace uso de este patrón en la aplicación, en el método *start* de la clase *ConfServerApp.h* al leer el script *configuracion.conf*.

Consideraciones parciales

Se definieron 8 historias de usuario, las cuales permitieron tener la base para la implementación de la aplicación. Los requisitos del sistema y la arquitectura propuesta permitieron obtener un diseño robusto de la aplicación. Además se aplicaron los patrones necesarios para lograr flexibilidad y eficiencia. Las definiciones de las tarjetas CRC aportaron en la identificación de las responsabilidades de las clases y las colaboraciones de las mismas.

Capítulo 3: “Implementación y pruebas”

3.1 Introducción

En el presente capítulo se identifican y describen las tareas de ingeniería y se llevan a cabo las pruebas de aceptación realizadas a cada una de las HU, con el objetivo de validar que el sistema cumpla con lo especificado por el cliente. También se realizan pruebas de rendimiento al producto obtenido, para analizar el impacto que causa en el módulo Configuración de manera general. Además se incluyen acápites que contendrán información relacionada con los estilos de codificación y documentación.

3.2 Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El plan de entrega está compuesto por iteraciones de no más de tres semanas. Al final de la última iteración el sistema estará listo para entrar en producción (10).

3.2.1 Tareas de ingeniería

Las tareas de ingeniería (TI) definen cada una de las actividades que dan cumplimiento a cada HU y se describen de forma clara; de manera tal que se entienda lo que el sistema tiene que hacer y facilite su construcción.

En la tabla 15 se ilustra la relación de las tareas por historias de usuario en sus correspondientes iteraciones.

Tabla 15: Relación HU-TI.

Iteración	Historia de usuario	Tareas por historias de usuario
1	**Generar trazas de tipo adición.**	1. Crear una clase encargada de representar una traza. 2. Implementar una función encargada de crear trazas de tipo adición.
	Generar trazas de tipo modificación.	1. Implementar una función encargada de crear trazas de tipo modificación.
	Generar trazas de tipo	1. Implementar una función encargada

	eliminación.	de crear trazas de tipo eliminación.
	Detectar cambios en la configuración del SCADA Galba.	1. Implementar un mecanismo que se encargue de identificar los cambios ocurridos en la configuración del SCADA.
2	**Enviar trazas hacia el módulo de histórico.**	1. Implementar una función encargada de enviar un listado de trazas hacia el módulo de histórico.
	Analizar la descripción de una traza.	1. Crear una clase encargada de manejar información referente a la descripción de una traza. 2. Implementar una función encargada de analizar la descripción de una traza.
3	**Imprimir trazas en consola.**	1. Modificar el script *configuración.conf* dando la posibilidad de establecer que el servidor imprima las trazas en la consola o no. 2. Implementar un algoritmo que permita mostrar las trazas generadas en la consola.
	Almacenar trazas en fichero.	1. Modificar el script *configuración.conf* dando la posibilidad de establecer que el servidor almacene las trazas en un fichero en el disco duro o no. 2. Implementar un algoritmo que permita almacenar las trazas generadas en un fichero en el disco duro.

A continuación se describen las correspondientes TI a las HU descritas en las tablas 16 a la 27. Para la confección de cada una de ellas se utilizaron tablas cuyo modelo cuenta con los siguientes campos:

- ✓ **No. de tarea:** Numeración continua que identifica a la tarea.
- ✓ **No. de HU:** Número de la HU a la cual pertenece.
- ✓ **Nombre de la tarea:** Identificación literal de la tarea.
- ✓ **Tipo de tarea:** Tipo de tarea, dígase diseño, desarrollo, prueba.
- ✓ **Puntos estimados:** Representación en porciento de la cantidad de tiempo estimada de una semana, que se utilizara para su realización.
- ✓ **Fecha inicio:** Fecha estimada de inicio de realización.
- ✓ **Fecha fin:** Fecha estimada de fin de realización.
- ✓ **Descripción:** Se describe en que consiste la tarea y que elementos deben cumplirse para declarar la tarea terminada.

Tabla 16: Tarea de ingeniería 1 de la HU 1.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU:** 1
Nombre de la tarea: Crear una clase encargada de representar una traza.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.4
Fecha inicio: 4/3/2015	**Fecha fin:** 6/3/2015
Descripción: Se realiza la implementación de una clase encargada de contener la información básica referente a una traza.	

Tabla 17: Tarea de ingeniería 2 de la HU 1.

Tarea de Ingeniería	
No. de tarea: 2	**No. de HU:** 1
Nombre de la tarea: Implementar una función encargada de crear trazas de tipo adición.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.6
Fecha inicio: 9/3/2015	**Fecha fin:** 11/3/2015

Descripción: Se realiza la implementación de una función encargada de crear trazas de tipo *adición*.

Tabla 18: Tarea de ingeniería 1 de la HU 2.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU:** 2
Nombre de la tarea: Implementar una función encargada de crear trazas de tipo modificación.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 23/3/2015	**Fecha fin:** 26/3/2015
Descripción: Se realiza la implementación de una función encargada de crear trazas de tipo *modificación*.	

Tabla 19: Tarea de ingeniería 1 de la HU 3.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU:** 3
Nombre de la tarea: Implementar una función encargada de crear trazas de tipo eliminación.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 6/4/2015	**Fecha fin:** 9/4/2015
Descripción: Se realiza la implementación de una función encargada de crear trazas de tipo *eliminación*.	

Tabla 20: Tarea de ingeniería 1 de la HU 4.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU:** 4
Nombre de la tarea: Implementar una función que se encargue de buscar los cambios ocurridos en la configuración.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 3
Fecha inicio: 10/4/2015	**Fecha fin:** 15/4/2015

Descripción: Se realiza la implementación de una función que realizará una comparación entre una configuración nueva cargada al SCADA y la configuración previamente existente, con el objetivo de detectar las modificaciones existentes.

Tabla 21: Tarea de ingeniería 1 de la HU 5.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU:** 5
Nombre de la tarea: Implementar una función encargada de enviar un listado de trazas hacia el módulo de histórico.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 17/04/2015	**Fecha fin:** 20/04/2015
Descripción: Se realiza la implementación de una función encargada de enviar, las trazas previamente generadas, hacia el módulo de histórico.	

Tabla 22: Tarea de ingeniería 1 de la HU 6.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU:** 6
Nombre de la tarea: Crear una clase encargada de manejar información referente a la descripción de una traza.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.3
Fecha inicio: 21/04/2015	**Fecha fin:** 23/04/2015
Descripción: Se realiza la implementación de una clase que modele y contenga la información de una traza.	

Tabla 23: Tarea de ingeniería 2 de la HU 6.

Tarea de Ingeniería	
No. de tarea: 2	**No. de HU:** 6
Nombre de la tarea: Implementar una función encargada de analizar la descripción de una traza.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.7
Fecha inicio: 24/04/2015	**Fecha fin:** 30/04/2015

Descripción: Se realiza la implementación de una función encargada de analizar la descripción de una traza, con el objetivo de desglosar esta información, que previamente se recibe en una cadena de texto, para hacer más sencillo el proceso de almacenamiento en la base de datos de histórico.

Tabla 24: Tarea de ingeniería 1 de la HU 7.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU:** 7
Nombre de la tarea: Modificar el script de configuración dando la posibilidad de establecer que el servidor imprima las trazas en la consola o no.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.5
Fecha inicio: 2/5/2015	**Fecha fin:** 2/5/2015
Descripción: Se realiza la modificación del script de configuración para que esté en condiciones de establecer el modo en que se tratarán las trazas de seguridad una vez que sean generadas.	

Tabla 25: Tarea de ingeniería 2 de la HU 7.

Tarea de Ingeniería	
No. de tarea: 2	**No. de HU:** 7
Nombre de la tarea: Implementar un algoritmo que permita imprimir las trazas generadas en la consola.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.5
Fecha inicio: 4/5/2015	**Fecha fin:** 5/5/2015
Descripción: Se implementa un algoritmo que verifique si se deben mostrar las trazas en la consola o no. Además de realizar esta acción en caso de que la verificación sea positiva.	

Tabla 26: Tarea de ingeniería 1 de la HU 8.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU:** 8
Nombre de la tarea: Modificar el script de configuración dando la posibilidad de	

establecer que el servidor almacene las trazas en un fichero en el disco duro o no.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.5
Fecha inicio: 6/5/2015	**Fecha fin:** 6/5/2015
Descripción: Se realiza la modificación del script de configuración para que esté en condiciones de establecer el modo en que se tratarán las trazas de seguridad una vez que sean generadas.	

Tabla 27: Tarea de ingeniería 2 de la HU 8.

Tarea de Ingeniería	
No. de tarea: 2	**No. de HU:** 8
Nombre de la tarea: Implementar un algoritmo que permita almacenar las trazas generadas en un fichero en el disco duro.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.5
Fecha inicio: 7/5/2015	**Fecha fin:** 8/5/2015
Descripción: Se implementa un algoritmo que verifique si las trazas se deben almacenar en un fichero en el disco duro. Además de realizar esta acción, en caso de que la verificación sea positiva.	

3.3 Estilos de codificación

Los estilos de codificación o programación definen la estructura y apariencia física del código, lo que facilita su comprensión, mantenimiento y lectura. Los estilos de codificación y documentación explicados en este capítulo son aplicados en el Centro de Informática Industrial, esto justifica el empleo de los mismos en el mecanismo para la generación de trazas a partir de los cambios ocurridos en la configuración del SCADA Galba. Los estilos utilizados evidencian la intención de lograr una aplicación que pueda ser comprendida sin necesidad de tener un amplio conocimiento respecto al tema.

3.3.1 Definición de clases

Las declaraciones de clases tienen su llave de apertura una línea más abajo de la declaración y el nombre de la clase comienza con mayúscula. Los nombres de las mismas son sustantivos

singulares, que deben reflejar qué hacen y no cómo lo hacen (28). En la siguiente figura se muestra un ejemplo.

```
class LogGenerator
{
public:
    LogGenerator();

    ...
}
```

Figura 5: Definición de clases.

3.3.2 Definición de métodos

Los nombres de los métodos son frases que incluyen verbos, dado que los mismos generalmente son el producto de concatenar varias palabras, se debe emplear la primera palabra en minúscula, mayúscula para denotar la letra de inicio de cada una de las palabras restantes por las que esté formado y minúscula para las letras intermedias. Los atributos pasados por parámetro se separarán por una coma y un espacio después de ésta en caso de existir más de uno (28).

3.3.3 Estructuras de control

Al hacer uso de las estructuras de control se deben emplear las letras i, j, k, l, m, p, q, r para contadores en ciclos. Emplear correctamente los tipos de ciclos: si es al menos una vez usar do-while, si es ninguna o más veces usar while-do, y si se conoce el número exacto de ciclos usar for (28). En la siguiente figura se muestra un ejemplo.

```
if ( pMask & SAVE_LOG )
{
    for ( uint_fast32_t i = 0; i < tSize; ++i )
    {
        if ( mLogs[i]->previousValue.size() || mLogs[i]->currentValue.size() )
        {
            tDescription = mLogs[i]->description + " Valor anterior: " +
                    mLogs[i]->previousValue + "; Valor actual: " +
                    mLogs[i]->currentValue + ";";
        }
        else
        {
            tDescription = mLogs[i]->description;
        }
        __LOG_INFO__(tDescription.c_str());
    }

}
```

Figura 6: Estructuras de control.

3.3.4 Codificación en general

De manera general, la codificación seguirá las siguientes pautas (28):

- ✓ Alinear secciones del código.
- ✓ Alinear verticalmente llaves de apertura y cierre.
- ✓ Usar espacios antes y después de los operadores que el lenguaje de programación permita.
- ✓ Emplear líneas en blanco para organizar el código, permitiendo crear párrafos de código para una mejor lectura.
- ✓ Evitar colocar más de una sentencia por línea.

3.4 Estilo de documentación

Se utilizó la herramienta Doxygen para generar la documentación del código de las distintas clases.

Se adopta el estilo de bloques de documentación JavaDoc, el cual consiste en un bloque de comentario de estilo C. Este bloque de comentario comienza con dos asteriscos, los asteriscos que se encuentran en la línea de la mitad son opcionales (28). Ejemplo:

```
/**
 * @brief Método encargado de parsear la descripción de una traza.
 * @param pLogDescription Cadena de texto con la descripción de la traza.
 * @author Jorge Alfonso Orama Pérez jaoramas@estudiantes.uci.cu.
 * @date 23/02/2015
 */
inline void Parser(std::string pLogDescription)
{
    ...
}
```

Figura 7: Estilos de documentación.

@**brief**: Indica una breve descripción de la parte del código que se está documentando.

@**param**: Brinda explicación de los parámetros en caso de que existan.

@**author**: Muestra el nombre y la dirección de contacto del programador.

@**date**: Indica la fecha de creación.

3.5 Diagrama de despliegue

Un diagrama de despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para modelar la disposición física de los artefactos software en nodos (usualmente plataforma de hardware) (29).

A continuación se muestra el diagrama de despliegue de la solución propuesta seguido de una breve explicación de los nodos y sus conexiones:

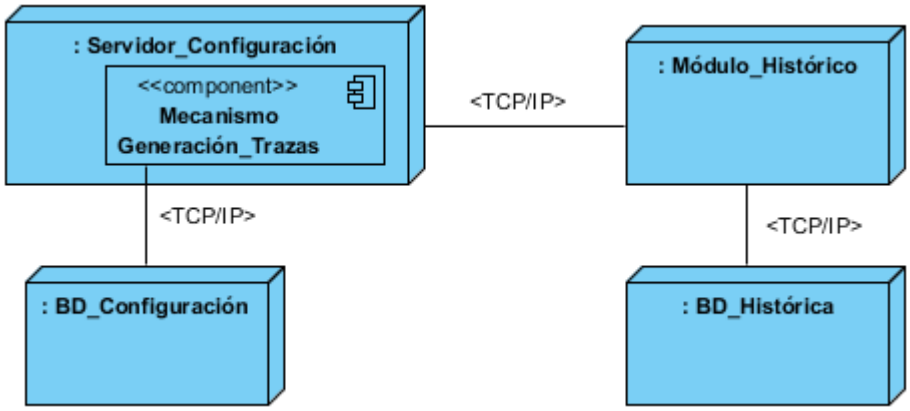


Figura 8: Diagrama de despliegue (elaboración propia).

- ✓ **BD_Configuración:** Nodo donde se encuentra la base de datos que almacena las configuraciones del SCADA Guardián del ALBA, la misma está soportada por el gestor PostgreSQL.

- ✓ **Servidor_Configuración:** Nodo que representa la PC donde se ejecuta el servidor de configuración del SCADA Galba. Dentro, se encuentra contenido el mecanismo de generación de trazas, el cual se conecta con el nodo BD_Configuración mediante el protocolo TCP/IP, haciendo uso de la biblioteca pqxx, con el objetivo de obtener la configuración actual y la configuración anterior del sistema, para generar información sobre los cambios que han ocurrido entre las mismas. El servidor de configuración se conecta con el módulo de histórico mediante el protocolo TCP/IP, haciendo uso de la capa de comunicación, con el objetivo de enviar la información de las trazas generadas por el mecanismo.

- ✓ **Módulo_Histórico:** Nodo que representa la PC donde se ejecuta el módulo de almacenamiento de datos históricos del Guardián del ALBA. Se conecta con el nodo BD_Histórica mediante al protocolo TCP/IP con el objetivo de hacer persistente las trazas de seguridad generadas.

- ✓ **BD_Histórica:** Este nodo representa la PC donde se encuentra la base de datos del módulo de histórico del Galba, la misma está soportada por el gestor PostgreSQL.

3.6 Pruebas

El desarrollo de sistemas de software implica una serie de actividades de producción en las que las posibilidades de que aparezca el fallo humano son grandes. Los errores pueden empezar a darse desde el primer momento del proceso, en el que los objetivos (…) pueden estar especificados de forma errónea o imperfecta, así como en posteriores pasos de diseño y desarrollo (…) Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software ha de ir acompañado de una actividad que garantice la calidad. Las pruebas del software son un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación (30).

3.6.1 Pruebas de rendimiento

Las pruebas de rendimiento tienen que diseñarse para asegurar que el sistema pueda procesar su carga esperada. Esto normalmente implica planificar una serie de pruebas en las que la carga se va incrementando regularmente hasta que el rendimiento del sistema se hace inaceptable. Como sucede con otros tipos de pruebas, las de rendimiento se ocupan tanto de demostrar que el sistema satisface sus requerimientos como de descubrir problemas y defectos en el sistema (31).

Las pruebas aplicadas a la solución propuesta se centraron en determinar el impacto que causa, en cuestiones de rendimiento, el mecanismo de generación de trazas del módulo Configuración en el Guardián de ALBA. Para ello se registraron mediciones del tiempo de ejecución, uso del CPU y consumo de memoria RAM, una vez que se generan cambios en configuraciones de 10000, 50000 y 100000 puntos. Las mediciones mencionadas fueron realizadas primeramente, con el mecanismo desactivado, es decir, sin generar trazas y en un segundo momento con el mismo en ejecución, posibilitando así realizar un análisis comparativo. Además de esto se fueron desactivando funcionalidades prescindibles del mecanismo para lograr un mejor rendimiento. Las pruebas fueron desarrolladas en una estación de trabajo que presentaba un procesador Intel Celeron a 2.6 GHz y una memoria RAM de 1GB de capacidad. A continuación se muestran las pruebas realizadas.

Tabla 28: Prueba de rendimiento 1.

Prueba de rendimiento
No. 1

Descripción: Prueba para analizar el rendimiento del módulo Configuración.		
Condiciones de la prueba: El mecanismo de generación de trazas estará desactivado.		
Tiempo de ejecución:	**Uso del CPU:**	**Uso de memoria RAM:**
3.779 s	4% - 7%	4.4 MB

Tabla 29: Prueba de rendimiento 2.

Prueba de rendimiento		
No. 2		
Descripción: Prueba para analizar el rendimiento del módulo Configuración en un cambio de configuración de 10000 puntos analógicos calculados.		
Condiciones de la prueba: El mecanismo de generación de trazas estará activado. Las trazas serán mostradas en la consola. Las trazas serán almacenadas en un fichero en el disco duro. Las trazas serán enviadas hacia el módulo de histórico.		
Tiempo de ejecución:	**Uso del CPU:**	**Uso de memoria RAM:**
6.109 s	30%	7.4 MB

Tabla 30: Prueba de rendimiento 3.

Prueba de rendimiento		
No. 3		
Descripción: Prueba para analizar el rendimiento del módulo Configuración en un cambio de configuración de 50000 puntos analógicos calculados.		
Condiciones de la prueba: El mecanismo de generación de trazas estará activado. Las trazas serán mostradas en la consola. Las trazas serán almacenadas en un fichero en el disco duro. Las trazas serán enviadas hacia el módulo de histórico.		
Tiempo de ejecución:	**Uso del CPU:**	**Uso de memoria RAM:**
28 s	69% - 81%	20 MB

Tabla 31: Prueba de rendimiento 4.

Prueba de rendimiento		
No. 4		
Descripción: Prueba para analizar el rendimiento del módulo Configuración en un cambio de configuración de 50000 puntos analógicos calculados.		
Condiciones de la prueba: El mecanismo de generación de trazas estará activado. No se mostrarán trazas en consola. Las trazas serán almacenadas en un fichero en el disco duro. Las trazas serán enviadas hacia el módulo de histórico.		
Tiempo de ejecución:	**Uso del CPU:**	**Uso de memoria RAM:**
6.737 s	45% - 50%	20 MB

Tabla 32: Prueba de rendimiento 5.

Prueba de rendimiento		
No. 5		
Descripción: Prueba para analizar el rendimiento del módulo Configuración en un cambio de configuración de 100000 puntos analógicos calculados.		
Condiciones de la prueba: El mecanismo de generación de trazas estará activado. No se mostrarán trazas en consola. Las trazas serán almacenadas en un fichero en el disco duro. Las trazas serán enviadas hacia el módulo de histórico.		
Tiempo de ejecución:	**Uso del CPU:**	**Uso de memoria RAM:**
13.840 s	80% - 83%	34 MB

Tabla 33: Prueba de rendimiento 6.

Prueba de rendimiento
No. 6
Descripción: Prueba para analizar el rendimiento del módulo Configuración en un cambio de configuración de 100000 puntos analógicos calculados.

Condiciones de la prueba: El mecanismo de generación de trazas estará activado. No se mostrarán trazas en consola. No se almacenarán trazas fichero. Las trazas serán enviadas hacia el módulo de histórico.		
Tiempo de ejecución:	**Uso del CPU:**	**Uso de memoria RAM:**
2.173 s	47% - 50%	32 MB

Producto del análisis realizado, se llegó a la conclusión de que el "Mecanismo de generación de trazas a partir de los cambios ocurridos en la configuración del SCADA Galba", no afecta la función principal de módulo Configuración, pero si afecta su rendimiento en sentido general, por lo que se proponen las siguientes medidas para su mejor ejecución:

1. La funcionalidad *mostrar trazas en consola,* es la que mayor impacto causa en el rendimiento del módulo Configuración, fundamentalmente en el tiempo de ejecución. Solo debe ser activada cuando se necesite realizar algún monitoreo de pequeños cambios en la configuración, y no se cuente con la posibilidad de ejecutar el monitor de servicios del SCADA Galba. En otro caso es ineficiente mantenerla activa.

2. La funcionalidad *imprimir trazas en un fichero*, causa también un imparto importante en el rendimiento del sistema. Por lo que es recomendable el uso de la misma solo cuando se desee realizar algún monitoreo de cambios de configuración con volúmenes menores de 50000 variables.

3. La funcionalidad *enviar trazas hacia histórico*, es la de mayor importancia dentro del mecanismo y se ejecuta en todo momento. La misma causa una carga permisible en el sistema.

4. Se recomienda que cuando el mecanismo de generación de trazas del módulo Configuración del SCADA Guardián del ALBA se encuentre operacional, no se activen las funcionalidades *mostrar trazas en consola* y *almacenar trazas en un fichero,* siempre y cuando no sean imprescindibles para alguna tarea específica que se quiera desarrollar en un momento determinado.

3.6.2 Pruebas de aceptación

Las pruebas de aceptación son diseñadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada. Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Así mismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación (25).

Como criterio de aprobación de cada caso de prueba, se determinó que la evaluación de la prueba sea *satisfactoria*. Esta evaluación se obtiene cuando no se detectan fallas críticas que interfieran en el funcionamiento del sistema, además de que el resultado esperado debe ser igual al obtenido.

Para representar las pruebas de aceptación se definieron los siguientes elementos:

- ✓ **Código:** Representa al caso de prueba, incluye el número de HU y de la prueba.
- ✓ **No. de HU:** Número de la HU a la cual pertenece.
- ✓ **Nombre:** Conforma el nombre del caso de prueba.
- ✓ **Descripción:** Descripción del caso de prueba.
- ✓ **Condiciones de ejecución:** Describe las condiciones que debe cumplir el sistema para realizar el caso de prueba.
- ✓ **Entrada/Pasos de ejecución:** Incluye las acciones que debe realizar el sistema para llevar a cabo la prueba.
- ✓ **Resultados esperados:** Descripción de la respuesta ideal que debe dar el sistema ante el caso de prueba.
- ✓ **Resultado obtenido:** Respuesta real del sistema después de realizar el caso de prueba.
- ✓ **Evaluación de la prueba:** Clasificación de la prueba en satisfactoria o insatisfactoria.

A continuación se muestran las pruebas de aceptación realizadas a las historias de usuario de la solución propuesta:

Tabla 34: Caso de prueba de aceptación HU1_P1.

Caso de prueba de aceptación	
Código: HU1_P1	**No. de HU:** 1

Nombre: Generar trazas de tipo adición.
Descripción: Prueba para la funcionalidad de generar trazas de tipo adición.
Condiciones de ejecución: El operador debe haber salvado una configuración en el servidor haciendo uso del Editor del HMI. En la configuración salvada, debe haberse añadido al menos un nuevo recurso.
Entradas/Pasos de ejecución: Crear una cadena de texto con la descripción de la traza, especificando que es de tipo *adición*. Crear un objeto de tipo DataLog. Añadir la nueva traza creada, al contenedor mLogs.
Resultado esperado: Se generan las trazas de tipo adición y se almacenan en un contenedor.
Resultado obtenido: El contenedor mLogs fue llenado correctamente con las trazas esperadas.
Evaluación de la prueba: Satisfactoria.

Tabla 35: Caso de prueba de aceptación HU2_P1.

Caso de prueba de aceptación	
Código: HU2_P1	**No. de HU:** 2
Nombre: Generar trazas de tipo modificación.	
Descripción: Prueba para la funcionalidad de generar trazas de tipo modificación.	
Condiciones de ejecución: El operador debe haber salvado una configuración en el servidor haciendo uso del Editor del HMI. En la configuración salvada, debe de haberse modificado un recurso existente.	
Entradas/Pasos de ejecución: Crear una cadena de texto con la descripción de la traza, especificando que es de tipo *modificación*. Crear un objeto de tipo DataLog.	

Añadir la nueva traza creada, al contenedor mLogs.
Resultado esperado: Se generan las trazas de tipo adición y se almacenan en un contenedor.
Resultado obtenido: El contenedor mLogs fue llenado correctamente con las trazas esperadas.
Evaluación de la prueba: Satisfactoria.

Tabla 36: Caso de prueba de aceptación HU3_P1.

Caso de prueba de aceptación	
Código: HU3_P1	**No. de HU:** 3
Nombre: Generar trazas de tipo eliminación.	
Descripción: Prueba para la funcionalidad de generar trazas de tipo eliminación.	
Condiciones de ejecución: El operador debe haber salvado una configuración en el servidor haciendo uso del Editor del HMI. En la configuración salvada, debe de haberse eliminado un recurso existente.	
Entradas/Pasos de ejecución: Crear una cadena de texto con la descripción de la traza, especificando que es de tipo *eliminación*. Crear un objeto de tipo DataLog. Añadir la nueva traza creada, al contenedor mLogs.	
Resultado esperado: Se generan las trazas de tipo adición y se almacenan en un contenedor.	
Resultado obtenido: El contenedor mLogs fue llenado correctamente con las trazas esperadas.	
Evaluación de la prueba: Satisfactoria.	

Tabla 37: Caso de prueba de aceptación HU4_P1.

Caso de prueba de aceptación	
Código: HU4_P1	**No. de HU:** 4

Nombre: Detectar cambios en diferentes configuraciones.
Descripción: Prueba para la funcionalidad de detectar cambios en diferentes configuraciones.
Condiciones de ejecución: El operador debe haber salvado una configuración en el servidor haciendo uso del Editor del HMI. En la configuración salvada, deben existir cambios.
Entradas/Pasos de ejecución: Obtener desde la base de datos, la configuración anterior y la configuración nueva. Realizar comparaciones entre ambas configuraciones para determinar las modificaciones que se hayan realizado.
Resultado esperado: Se identifican las diferencias entre ambas configuraciones.
Resultado obtenido: Se identificaron las diferencias entre ambas configuraciones.
Evaluación de la prueba: Satisfactoria.

Tabla 38: Caso de prueba de aceptación HU5_P1.

Caso de prueba de aceptación	
Código: HU5_P1	**No. de HU:** 5
Nombre: Enviar trazas hacia el módulo de histórico.	
Descripción: Prueba para la funcionalidad de enviar trazas hacia el módulo de histórico.	
Condiciones de ejecución: Se deben haber generado las trazas de seguridad.	
Entradas/Pasos de ejecución: Se envían las trazas hacia el módulo de histórico haciendo uso de la capa de comunicación.	
Resultado esperado: Se envían las trazas hacia el módulo de histórico y son almacenadas en el sistema persistente.	

Resultado obtenido: Se enviaron las trazas hacia el módulo de histórico y fueron almacenadas en el sistema persistente.
Evaluación de la prueba: Satisfactoria.

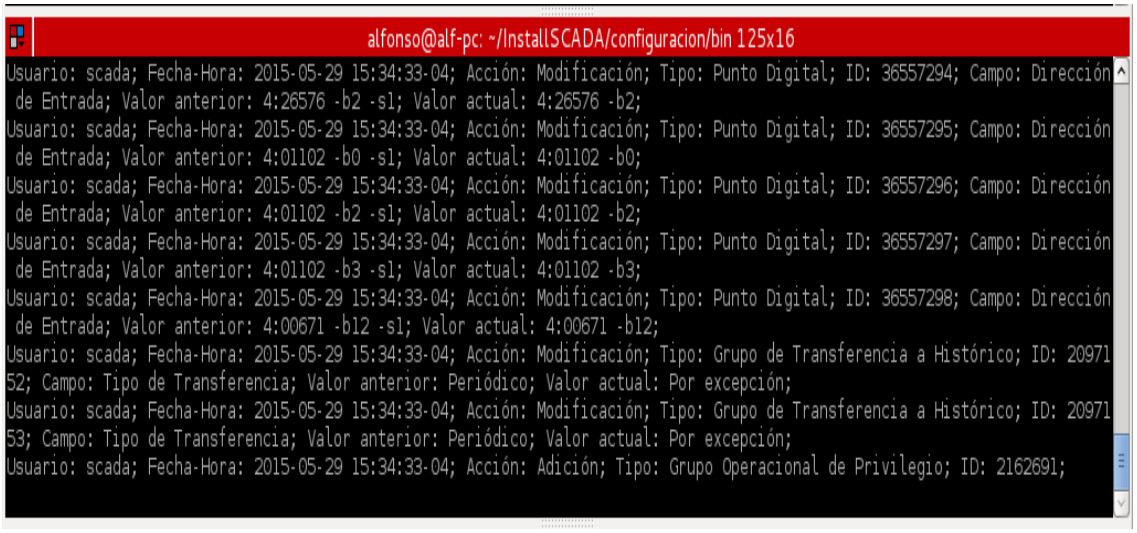
Tabla 39: Caso de prueba de aceptación HU6_P1.

Caso de prueba de aceptación	
Código: HU6_P1	**No. de HU:** 6
Nombre: Analizar la descripción de una traza.	
Descripción: Prueba para la funcionalidad de analizar la descripción de una traza.	
Condiciones de ejecución: Se deben haber generado las trazas de seguridad.	
Entradas/Pasos de ejecución: Crear un objeto de la clase ParserLogDescription.h. Introducir los datos de la descripción (un string) de una traza haciendo uso del objeto.	
Resultado esperado: Se desglosa la descripción y se almacenan los datos importantes en los atributos del objeto.	
Resultado obtenido: Se desglosa la descripción y se almacenan los datos importantes en los atributos del objeto.	
Evaluación de la prueba: Satisfactoria.	

Tabla 40: Caso de prueba de aceptación HU7_P1.

Caso de prueba de aceptación	
Código: HU7_P1	**No. de HU:** 7
Nombre: Imprimir trazas en consola.	
Descripción: Prueba para la funcionalidad de imprimir trazas en consola.	
Condiciones de ejecución: Se deben haber generado las trazas de seguridad. Debe de activarse en el script del servidor de configuración la opción que permite imprimir las trazas en la consola.	
Entradas/Pasos de ejecución: Recorrer el contenedor mLogs encargado de almacenar todas las trazas generadas	

para validar el tipo de traza. Imprimir en consola las trazas, en consecuencia con la validación realizada.
Resultado esperado: Se muestran en la consola la información referente a las trazas generadas.
Resultado obtenido:

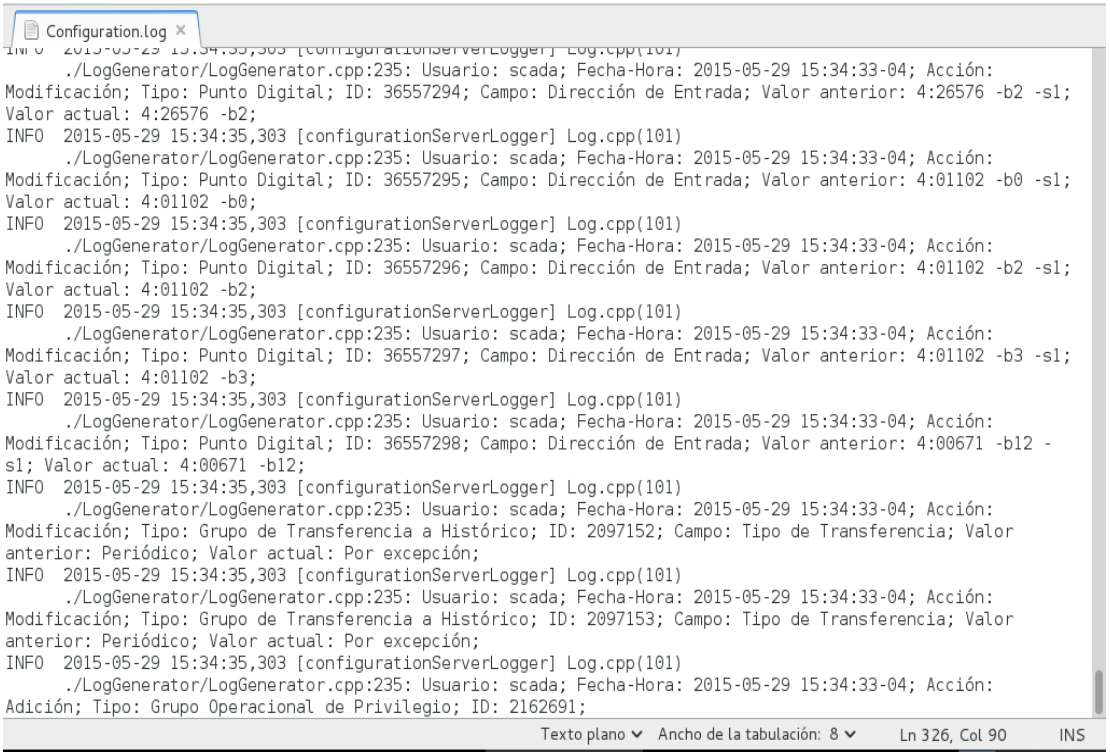


Evaluación de la prueba: Satisfactoria.

Tabla 41: Caso de prueba de aceptación HU8_P1.

Caso de prueba de aceptación	
Código: HU8_P1	**No. de HU:** 8
Nombre: Almacenar trazas en fichero.	
Descripción: Prueba para la funcionalidad de almacenar trazas en fichero.	
Condiciones de ejecución: Se deben haber generado las trazas de seguridad. Debe de activarse en el script del servidor de configuración la opción que permite almacenar las trazas en un fichero de texto en el disco duro.	
Entradas/Pasos de ejecución: Recorrer el contenedor mLogs encargado de almacenar todas las trazas generadas para validar el tipo de traza.	

Escribir en un fichero de texto las trazas, en consecuencia con la validación realizada.
Resultado esperado: Se almacena en un fichero de texto la información referente a las trazas de seguridad.
Resultado obtenido:



Evaluación de la prueba: Satisfactoria.

Las pruebas aplicadas al sistema fueron desarrolladas en 3 etapas. La primera etapa contó con 4 casos de prueba, los cuales fueron aplicados a las historias de usuario de prioridad alta, obteniéndose 3 casos de pruebas satisfactorios, representando un 75% de resultados satisfactorios en una primera iteración. Luego de corregir los errores encontrados, se realizó una segunda iteración de pruebas, alcanzando un 100% de resultados satisfactorios, consiguiéndose una primera versión del mecanismo.

La segunda etapa de pruebas contó con 2 casos de prueba, los cuales fueron aplicados a las historias de usuario de prioridad media en una primera iteración y obtuvieron una evaluación de satisfactorios, representando un 100% de resultados satisfactorios, permitiendo tener una segunda versión funcional de la aplicación.

La tercera etapa contó con 2 casos de prueba evaluando las historias de usuario de prioridad baja, los cuales fueron aplicados y obtuvieron una evaluación de satisfactorios en una primera iteración, representando un 100% de resultados satisfactorios.

El siguiente gráfico muestra la cantidad de pruebas de aceptación satisfactorias por iteraciones en cada etapa de pruebas.

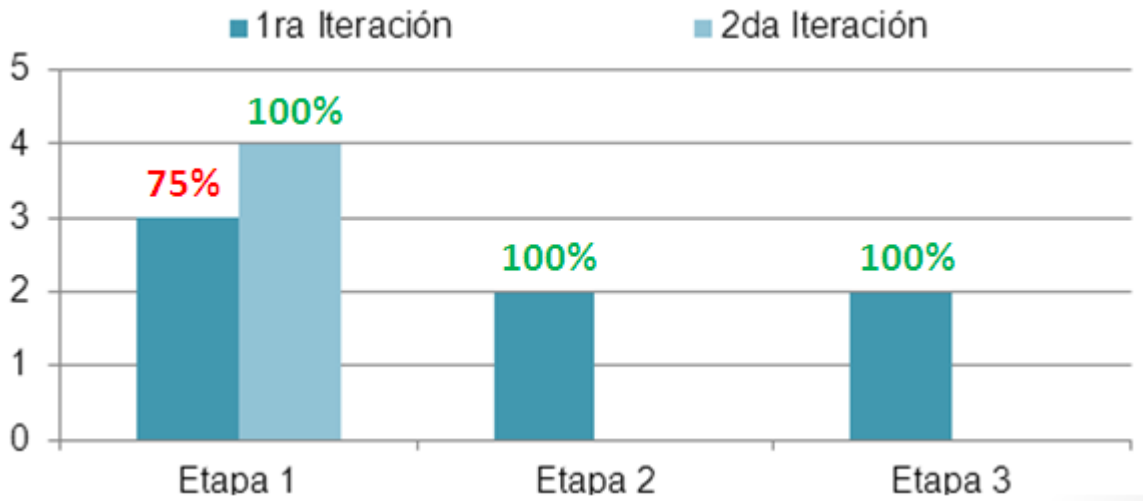


Figura 9: Pruebas de aceptación satisfactorias por iteraciones.

Al culminar con la aplicación de las pruebas y cumplir con el criterio de aprobación de las mismas, se termina la fase de pruebas, validando así que las historias de usuario de la solución propuesta cumplen con las especificaciones del cliente.

Consideraciones parciales

Al finalizar el presente capítulo, se definieron las tareas de ingeniería, las cuales evitaron una sobrecarga de trabajo, agilizando aún más el proceso. Se hizo una breve explicación de los principales estilos de codificación y documentación que guiaron el proceso de desarrollo. Se mostró y explicó el diagrama de despliegue da la solución. La aplicación de las pruebas de rendimiento permitieron analizar el impacto que causó el mecanismo dentro del módulo Configuración y la aplicación de las pruebas de aceptación, permitió comprobar la correcta implementación de las historias de usuarios definidas con anterioridad.

Conclusiones generales

Al término de la presente investigación, “Mecanismo para la generación de trazas a partir de los cambios ocurridos en la configuración del SCADA Galba”, se llegaron a las siguientes conclusiones:

- ✓ Se confeccionó el marco teórico de la investigación, posibilitando tener un punto de partida para el desarrollo del presente trabajo.
- ✓ Se diseñó e implementó un mecanismo para la generación de trazas a partir de los cambios ocurridos en la configuración del SCADA Galba, el cual permite tener en todo momento, un control estricto sobre los cambios ocurridos en la configuración del sistema, aportando la capacidad de auditar los mismos.
- ✓ Se documentó el mecanismo implementado siguiendo los estándares definidos por la herramienta Doxygen.
- ✓ Para asegurar que las historias de usuarios fueron implementadas correctamente y que la aplicación hace lo solicitado, se aplicaron, por parte del cliente, pruebas de aceptación, corrigiendo las no conformidades encontradas. Además se realizaron pruebas de rendimiento para evaluar el impacto del mecanismo dentro del módulo Configuración del Guardián del ALBA.

Recomendaciones

Una vez concluida la presente investigación se recomienda:

1. Incluir un nuevo elemento en la sintaxis de la traza que se genera, el cual hará referencia a la cantidad de bytes resultantes de la transferencia de información hacia el módulo de Histórico, siguiendo el formato estándar propuesto por el W3C.
2. Hacer un estudio sobre una posible optimización del mecanismo, con el objetivo de disminuir tanto como sea posible el consumo de recursos.

Referencias bibliográficas

1. **CEDIN, Consejo de dirección.** *Manual Organizacional. Centro de Informática Industrial (CEDIN).* La Habana, Cuba : s.n., 2012.

2. **HERNÁNDEZ HERNÁNDEZ, M.sc. Yaneisy y HERRERA VÁZQUEZ, Msc. Moisés.** *Especificación técnica de la interfaz de administración y auditoría de módulos y servicios.* La Habana : s.n., 2012.

3. **CORRALES PAUCAR, Dr. Luis.** *Interfaces de Comunicación Industrial.* Quito : s.n., 2007.

4. **BOYER, Stuart A.** *SCADA: supervisory control and data acquisition.* s.l. : International Society of Automation, 2009.

5. **ORDONEZ, Gabriel.** oocities.org. [En línea] 10 de 2009. [Citado el: 04 de 06 de 2015.] http://www.oocities.org/gabrielordonez_ve/FUNCIONES_DEL_SISTEMA_SCADA.htm.

6. **HERRERA VÁZQUEZ, Msc. Moisés.** *Introducción a la Arquitectura del Guardián del ALBA.* La Habana, Cuba : s.n., 2008.

7. **DE LA ROSA RODRÍGUEZ, Ing. Argelio, ESQUIJARROSA VALDÉS, Ing. Mayliuvis y BEYRIS SOULARY, Ing. Liliam Celia.** *Informe de análisis y diseño del Módulo Configuración. Proyecto Sistema de Supervisión y Control Guardián del ALBA.* 2013.

8. **HAP Group.** CIO America Latina. [En línea] [Citado el: 20 de 01 de 2015.] http://cioal.gpcinc.mx/2009/04/26/logrhythm-es-reconocido-como-la-mejor-solucion-para-incidentes-de-seguridad/.

9. **Splunk Inc.** Splunk. [En línea] [Citado el: 20 de Enero de 2015.] http://www.splunk.com/.

10. **PENADÉS, Carmen y LETELIER TORRES, Patricio Orlanndo.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Valencia : s.n., 2006. ISSN-e 1666-1680.

11. **SANTIAGO, María Lourdes.** *Proceso de Desarrollo Unificado (RUP).* 2009.

12. **LOAIZA, Alfredo Douglas.** Ingeniería de Software: Metodología XP. [En línea] [Citado el: 04 de 02 de 2015.] http://pnfiingenieriadesoftwaregrupocuatro.blogspot.com/2012/07/bienvenidos-al-blog.html.

13. **JOSKOWICZ, José.** *Reglas y prácticas en eXtreme Programming.* Vigo, España : s.n., 2008.

14. **ACEBAL, César F. y CUEVA LOVELLE, Juan M.** *Extreme Programming (XP): un nuevo método de desarrollo de software.* Oviedo, España : Novatica, 2002. págs. 8-12.

15. **RUMBAUGH, James, BOOCH, Grady y JACOBSON, Ivar.** *El lenguaje unificado de modelado: manual de referencia.* s.l. : ADDISON-WESLEY, 2007. 978-84-7829-087-1.

16. **LAFUENTE, Guillermo Javier.** UML Unified Modeling Languaje. [En línea] 2007. [Citado el: 10 de Febrero de 2015.] http://gidis.ing.unlpam.edu.ar/personas/glafuente/uml/uml.html.

17. **STROUSTRUP, Bjarne.** *El lenguaje de programación C++.* Wilmington, USA : Addison-Wesley Iberoamericana, S.A., 1993. ISBN 0-201-60104-4.

18. **ALVAREZ, Sara.** desarrolloweb.com. [En línea] Julio de 2007. [Citado el: 12 de Diciembre de 2014.] http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

19. **The PostgreSQL Global Development Group.** Sitio oficial de PostgresSQL. [En línea] [Citado el: 2 de Diciembre de 2014.] http://www.postgresql.org.es/sobre_postgresql.

20. **KORRY DOUGLAS, S. D.** *PostgreSQL.* s.l. : Sams Publishing, 2003. pág. 816.

21. **Oracle Corporation.** Netbeans Official Site. [En línea] [Citado el: 10 de Diciembre de 2014.] http://netbeans.org/features/index.html.

22. **MORENO CECA, JESÚS.** *Nuevas tecnologías de la manipulación usando sensorización integrada.* 2014. Tesis Doctoral.

23. **HERNANDIS, J. A.** Why Visual Paradigm for UML? [En línea] Septiembre de 2010. [Citado el: 13 de Enero de 2015.] http://www.visual-paradigm.com/product/vpuml.

24. **The Apache Software Foundation.** The Apache Software Foundation: Sitio Oficial. [En línea] [Citado el: 07 de Junio de 2015.] https://logging.apache.org/log4cxx/.

25. **Springer-Verlag.** *Guía avanzada de gestión de requisitos.* 2008.

26. **BECK, Kent.** *Extreme Programming Explained: embrace change.* s.l. : Addison-Wesley Professional, 2000.

27. **LARMAN, Craig.** *UML y Patrones.* s.l. : Pearson, 1999. págs. 185 - 215.

28. **CHÁVEZ LORENZO, Ariel y GARCÍA HERNÁNDEZ, Luis E.** *Estándares de codificación para C++. Proyecto SCADA Guardián del ALBA.* La Habana : s.n., 2010.

29. **OMG.** *OMG, OMG. Unified Modeling Language (OMG UML), Superstructure V2. 1.2. The Object Management Group (OMG).* MA, USA : Needham, 2007. pág. 202.

30. **PRESSMAN, Roger S.** *Ingeniería del Software: Un enfoque práctico.* 6. s.l. : Interamericana, 2006.

31. **SOMMERVILLE, Ian y GALIPIENSO, María Isabel Alfonso.** *Ingeniería del software.* Madrid : Pearson Educación, 2005.

32. **CAMPO, Gustavo Damián.** *Patrones de Diseño, Refactorización y Antipatrones. Ventajas y Desventajas de su Utilización en el Software Orientado a Objetos.* 2009.