Universidad de las Ciencias Informáticas

Facultad 5

Título: Notificación de alarmas por mensajes de texto para el SCADA UX.

Trabajo de Diploma para optar por el título de

Ingeniero en ciencias Informáticas

Autor: Liset Antunez Ojeda

Tutores: Luis Ángel Ravelo Hernández

Yaima Antunez Ojeda

Consultante: Antonio Cedeño Pozo

Junio 2012

DECLARACIÓN DE AUTORÍA

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

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

Autora: _____
Liset Antunez Ojeda

Tutor: _____
Ing. Luis Ángel Ravelo Hernández

DATOS DE CONTACTO

Ing. Luis Ángel Ravelo Hernández

Graduado de Ingeniero en Ciencias Informáticas en el año 2009 con seis años de experiencia en el desarrollo de software, específicamente en el desarrollo de subsistemas de comunicación para sistemas distribuidos. Se encuentra vinculado al departamento de Componentes del Centro de Desarrollo de Informática Industrial (CEDIN).

Correo: laravelo@uci.cu

Ing. Yaima Antunez Ojeda

Graduada de Ingeniera en Ciencias Informáticas en el año 2008 y profesora instructora con cuatro años de experiencia docente en la Universidad de Ciencias Informáticas y seis en la producción de software.

Correo: yantunez@uci.cu

Ing. Antonio Cedeño Pozo

Graduado de Ingeniero en Ciencias Informáticas de la Universidad de las Ciencias Informáticas en el 2009. Seis años de experiencia en el desarrollo de software.

Correo: acedeno@uci.cu

AGRADECIMIENTOS

Agradezco a mi mamá y mi papá porque han sabido vivir para sus hijas, y lo han dado todo por nosotras, por el apoyo incondicional que nos han dado siempre, por tanto amor y dedicación, por estar ahí cada vez que los necesitamos, por ser tan lindos educándonos y aconsejándonos, simplemente les doy las gracias por existir y por ser mis padres.

A mi hermana Yaima, por ser esa segunda madre que cualquiera quisiera tener, siendo siempre mi guía, apoyándome en todo, trazándome las metas más altas que pueda alcanzar, gracias por dejarme contar contigo siempre.

A Elena mi abuela, mi tía Chela, a Tita, mi abuelo Valdito, mis primas, en general a toda mi familia por estar siempre pendientes de mí, y por preocuparse y apoyarme hasta el final de mi carrera.

A Luis Ángel que más que tutor y cuñado se convirtió en ese hermano mayor que nunca tuve, y me ayudó mucho en estos cinco años.

A Batista, un agradecimiento especial, por haberme soportado estos años como un amigo incondicional.

A mis más que amigos Irina, Heidy, Rubino, Angélica, Amalia, Evián, Tony, que fueron mis padrinos este último año, y más que amigos son parte de la familia.

A mis amistades…que son muchas…las de la UCI que vivieron conmigo aquí al diario, principalmente las chicas de mis apartamentos, que fueron varios, pero que pasamos muchísimas cosas juntas, buenas y malas.

En fin a todas aquellas personas que se preocuparon por mí… A todos Gracias… Liset

DEDICATORIA

Dedico mi tesis:

A mi mamá y mi papá que son lo más importante que tengo y que son los mejores padres del mundo.

A mi hermana Yaima, que siempre ha sido mi guía y apoyo.

A mis abuelos paternos que estoy segura que se hubieran sentido orgullosos de mí.

RESUMEN

Los sistemas SCADA se definen como aplicaciones diseñadas para controlar a través de un ordenador y con dispositivos de campo, las operaciones de control, supervisión y registro de datos de cualquier operación industrial. La Universidad de las Ciencias Informáticas (UCI) actualmente desarrolla el SCADA UX como un producto 100% cubano, para la supervisión y control de procesos industriales, que puede ser aplicado a cualquier proceso que requiera automatización. Entre las funcionalidades de estos sistemas se encuentra la notificación de alarmas, para avisar la existencia de alguna condición anormal en el sistema. Algunos sistemas SCADA han estado sin cubrir la necesidad de comunicar de manera inmediata, o en el menor tiempo posible, los errores del sistema de producción.

El sistema que se propone a continuación permitirá al SCADA UX una inmediata notificación de las alarmas, por mensaje de texto a través de radios TETRA a los operadores encargados de solucionar el problema que generó dicha alarma, aprovechando las ventajas que ofrece este estándar, proporcionándole al SCADA UX mayor privacidad y confidencialidad.

PALABRAS CLAVE

Alarmas, radios, SCADA, TETRA.

TABLA DE CONTENIDOS

Índice de Figuras.

Índice de Tablas.

INTRODUCCIÓN

La informática y las comunicaciones están consideradas en todo el mundo, como uno de los factores estratégicos y de supervivencia más importantes para cualquier organización. Aspectos como la información en línea, la productividad, la calidad y la eficiencia en los procesos y servicios que se implementan, basados en las tecnologías de información y comunicaciones, son aspectos cruciales, tanto por su impacto, como por la exigencia, cada vez mayor, del mundo globalizado. Estas dos nuevas tecnologías soportan, hoy en día, todos los procesos críticos de las organizaciones modernas. El impacto de estas dos áreas, la informática y las comunicaciones, ha sido definitivo en la transformación de la sociedad industrial de los siglos pasados, en la nueva sociedad de la información y del conocimiento del siglo XXI.

El rápido desarrollo de las ciencias y la tecnología ha llevado a la sociedad a entrar en un nuevo milenio inmerso en lo que se ha dado llamar “Era de la informatización”. Cuba ha sido uno de los países en desarrollo que más ha avanzado en potenciar el uso de las nuevas tecnologías. Actualmente se sigue perfeccionando el trabajo y ampliando el radio de acción de estas en beneficio de todas las personas. Se planifican metas ambiciosas que están a la altura de los países del primer mundo y que ya hoy no se está muy lejos de poderlas alcanzar, pues se cuenta con la Industria Cubana del Software, que con la participación de la Universidad de las Ciencias Informáticas (UCI) y otras empresas productoras de software del país se están dando grandes pasos.

La Universidad de las Ciencias Informáticas (UCI), es una universidad productiva, cuya misión es producir software y servicios informáticos a partir de la vinculación estudio–trabajo como modelo de formación, basada en la integración de los procesos de formación, investigación y producción en torno a una temática para convertirla en una rama productiva. La universidad cuenta con una serie de proyectos productivos que se encuentran organizados en centros de desarrollo, entre los cuales está el Centro de Informática Industrial (CEDIN), centro que desarrolla productos y servicios informáticos de automatización industrial y computación gráfica, con un alto valor agregado y que cumplan las necesidades y expectativas de los clientes, potenciando la formación especializada y la investigación. El centro está dividido en líneas de desarrollo de software entre las cuales se encuentran: Visualización y Realidad Virtual, y Automática Aplicada.

Uno de los proyectos que actualmente se desarrolla en la línea de Automática Aplicada, es el TETSCADA (Interfaz SCADA (*Supervisory Control and Data Adquisition*) para acceso a radio TETRA (*Terrestrial Trunked Radio*, Radio Troncalizado Terrestre)), el cual proporciona un dispositivo pasarela (*Gateway*) que sirve como transporte entre los equipos industriales y las aplicaciones de gestión SCADA. El dispositivo se conecta con los terminales de radio TETRA (interfaz RS-232) y con los dispositivos industriales (interfaces RS-232, RS-485 y Ethernet) y es gestionado por un firmware que implementa los mecanismos de comunicación de dichas interfaces. TETRA es un estándar global de comunicación de radio desarrollado para satisfacer las necesidades de los clientes que necesitan rápida conexión punto a punto y punto a múltiples puntos vía radio.

La línea de Automática Aplicada se especializa fundamentalmente en el desarrollo de sistemas SCADA, este se define como una aplicación diseñada para controlar a través de un ordenador y con dispositivos de campo, cualquier operación industrial. Entre las funcionalidades más comunes y que no deben faltar en estos sistemas se encuentra la notificación de alarmas en determinadas situaciones. Las alarmas son avisos del sistema que detectan la presencia de una condición anormal y notifican dicha situación al componente visual del SCADA, de manera que el operador pueda reconocerla y ejecutar las acciones correctivas pertinentes.

Actualmente se encuentra en desarrollo el proyecto SCADA UX, como un producto genérico del CEDIN, para la supervisión y control de procesos industriales, el cual entre sus funcionalidades permite el envío de las alarmas desde el módulo de procesamiento y a través del middleware hacia el HMI (Interfaz hombre-máquina). El sistema de notificación de alarmas existente no permite la interacción directa del sistema con los operadores que se encuentran en el campo, lo que trae como consecuencia que no tengan notificación inmediata del sistema cuando ocurre una alarma que ellos puedan solucionar, lo que implica que los operadores que se encuentran frente a las consolas deban localizarlos y hacer la notificación, provocando la pérdida de tiempo muy valioso en situaciones extremas.

Por la situación antes expuesta surge como **problema científico**: ¿Cómo lograr la notificación de alarmas por mensajes de texto en el SCADA UX a través de redes TETRA?

Para dar solución a este problema se define como **objetivo general** de la investigación: Desarrollar un subsistema de notificación de alarmas por mensajes de texto para el SCADA UX utilizando redes TETRA.

Para darle cumplimiento a dicho objetivo se planteó como **objeto de estudio** la notificación de alarmas en

sistemas SCADA, y como **campo de acción**, la notificación de alarmas en sistemas SCADA usando redes TETRA.

Como **idea a defender** para la realización de la investigación se toma como base la premisa que con la implementación del subsistema de notificación de alarmas por mensajes de texto para el SCADA UX utilizando redes TETRA, se tendrá un sistema más eficiente y confiable para los operadores.

Para cumplir con el objetivo trazado y resolver el problema planteado, se proponen las siguientes **tareas de investigación**:

1. Estudio del estado del arte de la notificación de alarmas en sistemas SCADA, así como del estándar TETRA profundizando en sus funcionalidades para el envío de mensajes para una mejor comprensión del mismo.
2. Estudio de las herramientas y tecnologías a utilizar para facilitar el trabajo.
3. Identificación de los requerimientos de sistema para definir su futura implementación.
4. Definición de la arquitectura del sistema para lograr que este sea robusto.
5. Diseño de la aplicación.
6. Implementación de las funcionalidades identificadas.
7. Realización de pruebas unitarias al sistema para validar la solución.

Para la realización de las tareas de investigación se han empleado los **métodos de investigación** que se describen a continuación.

Métodos Teóricos:

Entre estos se empleó:

- **Analítico-Sintético,** con el objetivo de analizar y aumentar los conocimientos entorno al objeto de estudio a partir de consultar la bibliografía científica correspondiente, para después, haciendo uso de la síntesis, lograr resumir y exponer los resultados obtenidos del análisis.
- **Histórico-Lógico**, para conocer con mayor profundidad los antecedentes y las tendencias actuales relacionadas con el objeto de estudio.
- **Modelación**, permite realizar abstracciones con el objetivo de explicar la realidad y encontrar la posible solución del problema planteado.

Métodos Empíricos: Permiten la observación y el análisis inicial de la información.

Entre estos se empleó:

- **Observación**, permite adquirir la información necesaria, y puede utilizarse en cualquiera de las fases de la investigación, además ofrece gran acercamiento a la realidad y permite ver la posible solución del problema desde diferentes ángulos.

El contenido de la investigación queda estructurado de la siguiente manera:

CAPÍTULO 1. Fundamentación teórica de la solución: En este capítulo se analizan los temas principales relacionados con el problema antes planteado. Se resume de forma general las características principales tanto del estándar TETRA como de los sistemas SCADA para una posible solución final.

CAPÍTULO 2. Tecnologías y herramientas: En este capítulo se describen las tecnologías y herramientas a utilizar para dar solución al problema planteado, desde herramientas para el diseño hasta la implementación.

CAPÍTULO 3. Presentación y diseño de la solución: En este capítulo se presentan los requisitos funcionales y no funcionales que debe tener el software y se muestra la concepción del sistema a partir del diagrama de casos de uso y las descripciones de los mismos; se desarrolla el diseño de la propuesta partiendo de la explicación de los patrones de diseño que se utilizan para modelar el software, se propone la arquitectura para guiar la realización de los principales aspectos del sistema.

CAPÍTULO 4. Implementación y pruebas de la solución: En este capítulo se describe la fase de implementación, se presenta el diagrama de componentes, además de describir las pruebas funcionales realizadas al sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción.

En este capítulo se abordarán aspectos fundamentales de los sistemas SCADA, centrándose fundamentalmente en la notificación de alarmas. Se hace un estudio de algunos estándares de comunicación enfatizando en el estándar TETRA, sobre todo en sus servicios, y esencialmente en los servicios de envío de mensajes de texto.

1.2 Sistemas SCADA.

SCADA de las siglas "*Supervisory Control And Data Adquisition*", es decir: sistema para la adquisición, supervisión y control de datos. Se trata de una aplicación de software especialmente diseñada para funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, etc.) y controlando el proceso de forma automática desde la pantalla del ordenador. Además, provee de toda la información que se genera en el proceso productivo a diversos usuarios, tanto del mismo nivel como de otros supervisores dentro de la empresa: control de calidad, supervisión, mantenimiento, etc. (1)

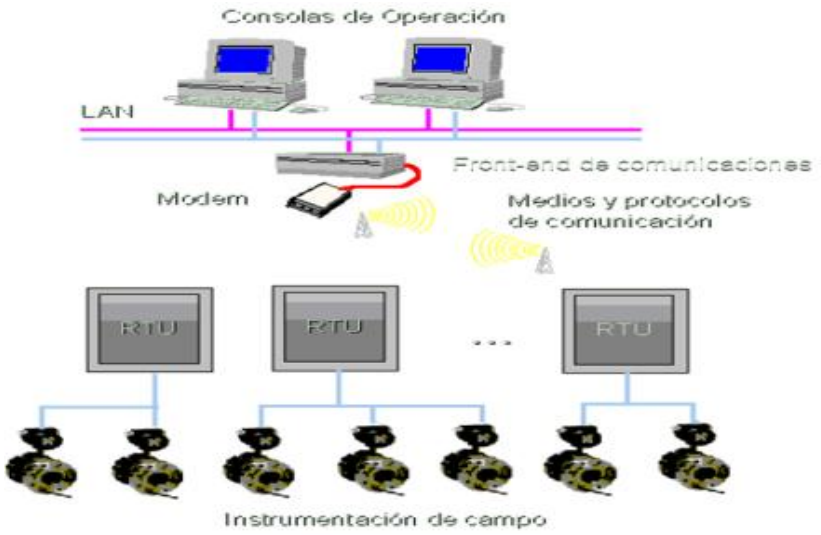

Fig. 1 Ejemplo de un SCADA.

Los módulos que permiten las actividades de adquisición, supervisión y control son los siguientes:

- Configuración: permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar.
- Interfaz Gráfica del Operador: proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado en el SCADA o importados desde otra aplicación durante la configuración del paquete.
- Módulo de proceso: ejecuta las acciones de mando preprogramadas a partir de los valores actuales de variables leídas.
- Gestión y archivo de datos: se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.
- Comunicaciones: se encarga de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA, y entre ésta y el resto de elementos informáticos de gestión. (1)

1.2.1 Funcionalidades Principales.

A continuación se muestran algunas de estas funcionalidades:

- **Adquisición y almacenamiento de datos:** Para la recolección y procesamiento de los datos en tiempo real y el almacenamiento de la información recibida, en forma continua y confiable para posteriores análisis.
- **Supervisión de procesos:** Para el monitoreo del funcionamiento del proceso, procesamiento estadístico de los datos y confección de reportes. Además se brindan notificaciones al operador del sistema sobre cambios detectados en la instalación. Estas notificaciones se clasifican principalmente en alarmas y eventos. Las **alarmas** se basan en la vigilancia de los parámetros de las variables del sistema. Son los sucesos no deseables, porque su aparición puede dar lugar a problemas de funcionamiento. Este tipo de sucesos requieren de la atención de un operador para su solución antes de que se llegue a una situación crítica que detenga el proceso. El resto de las situaciones normales, tales como puesta en marcha, paro, cambios de consignas de funcionamiento, consultas de datos, entre otras, serán los denominados **eventos** del sistema o sucesos. Los eventos no requieren de la atención del operador del sistema, registran de forma automática todo lo que ocurre en el sistema. También será posible guardar estos datos para su posterior consulta.

- **Control de procesos:** Para el control de los procesos de la fábrica, planta o industria, actuando sobre los reguladores autónomos básicos como eventos y alarmas, o directamente sobre el proceso mediante las salidas conectadas.
- **Transmisión de datos:** Para la transmisión de información entre dispositivos de campo y computadoras de control o entre dispositivos ubicados a un mismo nivel.
- **Presentación de la información:** Para la representación gráfica de los datos. Interfaz del Operador o HMI (*Human Machine Interface*). (2)

1.2.2 Las alarmas.

Alarmas

Son funcionalidades del sistema que detectan la presencia de una condición anormal y notifican dicha situación al componente visual del SCADA, de manera que el operador pueda reconocerla y ejecutar las acciones correctivas pertinentes.

Sumario de alarmas.

El sumario de alarmas es una interfaz gráfica que concentra las condiciones de procesos críticos, medios y bajos presentes en el sistema y cuyo objetivo primordial es guiar al operador a la detección del origen de la falla y supervisar la ejecución de las medidas de corrección automatizada o manual.

Principio de funcionamiento del sumario.

Los tipos de alarmas incluyen:

Momentánea: Alarma que desaparece del sumario, al retornar la variable al estado normal independientemente de si ha sido reconocida o no.

Mantenida: Alarma que desaparece del sumario, solo cuando la variable retorna al estado normal y la misma ha sido reconocida.

Nivel de severidad.

La jerarquía de la alarma se define por su nivel de severidad, lo cual determina su tratamiento, según lo siguiente:

Severidad 1 (Alarmas Críticas).

Son aquellas que requieren acción inmediata del operador:

1. Después que la acción correctiva automática ha sido activada.

2. Cuando la acción correctiva automática no está disponible.

En ambos casos el retardo o el fracaso de la acción correctiva pueden causar heridas al personal, producir daños ambientales o afectar las instalaciones, dañar o causar otras pérdidas sustanciales.

Severidad 2 (Media).

El objetivo de estas alarmas es advertir al operador que es necesario tomar medidas a fin de eliminar la desviación que activó la alarma y evitar la ejecución de alguna acción correctiva automática o manual. Un ejemplo típico de esto son las “pre alarmas”.

Severidad 3 (Advertencia de Alarmas).

Usado para objetivos de información relacionados con desviaciones de proceso, equipo de proceso estado anormal, estado de cierre no incluido en prioridad 1 ó 2, alarmas de sistema, etc.

Tratamiento de la alarma según tipo.

Alarmas de tipo Momentánea:

Este tipo de alarmas, suceden en las entidades en las cuales han sido configuradas o son establecidas previamente por diseño en el proyecto, y se mantienen en sumario mientras exista la condición anormal, desapareciendo del mismo al restablecerse la condición normal de la entidad. El reconocimiento o no de este tipo de alarmas no tiene influencia directa en su permanencia dentro del sumario.

Alarmas clasificadas dentro de este tipo:

- Alarmas de Falla de Comunicación en dispositivos o sub-canales.
- Alarmas de Falla de Instrumento.
- Alarma de Falla de no Variación en el Tiempo.

Alarmas de tipo Mantenida:

Este tipo de alarmas, suceden en las entidades en las cuales han sido configuradas, o son establecidas previamente por diseño en el proyecto, y su permanencia dentro del sumario está determinada por la existencia de dos condiciones: la persistencia del estado anormal y el reconocimiento de la misma. La alarma sólo puede borrarse del sumario al restablecerse la condición normal y ser reconocida por el operador del sistema.

Alarmas clasificadas dentro de este tipo:

- Alarmas de Falla de Ejecución de Comando.
- Alarmas de Cambio de Estado no Comandado.
- Alarmas de Tasa de Cambio.

- Alarmas de Cambio de Estado.
- Alarmas de Nivel.
- Alarma de Desviación. (3)

1.2.3 Notificación de alarmas.

Todo SCADA proporciona un sistema de notificación para informar al operador de las condiciones del proceso y del sistema. Este sistema permite la visualización, registro e impresión de alarmas de proceso y eventos del sistema. Existen dos sistemas de alarmas: local y distribuida. El sistema local se utiliza para mostrar y reconocer alarmas del dispositivo local conectado al SCADA. El sistema distribuido se utiliza para mostrar y reconocer alarmas de cualquier dispositivo, cuando el SCADA está conectado a un sistema en Red (mediante un Bus de datos).

Para visualizar las alarmas es preciso disponer de una interfaz gráfica en la cual, cuando se active la alarma, aparecerá toda la información relativa a la misma (hora y fecha, tipo de alarma, nombre, grupo, valores limites, etc.). Será necesario disponer de pulsadores de “recibida” para que el color del texto cambie indicando dicho reconocimiento de la alarma. Cuando se normalice el estado el mensaje dejará de visualizarse. Además de mostrarse las alarmas su ocurrencia se almacena en la base de datos para posteriores análisis.

Modos de presentación.

El sumario es representado como una lista de alarmas en modo contraído o expandido.

Sumario en Modo de lista contraída.

Muestra las cinco últimas alarmas independientemente del orden presentado en el sistema y ofrece barra de desplazamiento para visualizar las alarmas en su orden de aparición.

Sumario en Modo de lista expandida.

Representa las alarmas ordenadas de acuerdo con la relación severidad, prioridad, reconocimiento.

1.3 SCADA UX.

Actualmente en el CEDIN de desarrolla el SCADA UX, un sistema basado en el software libre, con nuevas tecnologías, con una arquitectura abierta capaz de crecer o adaptarse según las necesidades de las empresas que lo necesite, que permita al CEDIN contar con oportunidades de negocios, insertarse en nuevos mercados y la implementación y comercialización dentro y fuera del país.

Además es un producto genérico que puede ser aplicado a cualquier proceso que requiera automatización, con el mínimo de cambios a realizar. Se desarrollan además los módulos reutilizables, de forma que puedan ser utilizados para otras implementaciones, se realizan personalizaciones para distintas industrias y ofrece servicios asociados; además se puede integrar con otros sistemas. Permitiendo al CEDIN contar con un producto que podrá ser comerciable en nuevos mercados y áreas de negocios afines.

El SCADA UX también está conformado por varios módulos, que se presentan a continuación:

- **Drivers:** En este módulo se encuentran los controladores que permiten al sistema operativo interactuar, controlar y comunicarse con un dispositivo en particular, posibilitando así la transmisión de datos entre redes de computadoras.
- **Procesamiento y Recolector:** Es un framework en desarrollo de estructura modular que trabaja orientado a plugin, para garantizar la ejecución en tiempo real y se encarga de la planificación de los procesos de lectura y escritura sobre los dispositivos dentro del sistema, actúa realizando diversas funciones como la conversión de datos para su mejor asimilación y posterior uso. También posee conexión con la capa Middleware.
- **BDH:** Es el módulo encargado de manejar de forma clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante y se enviarán al HMI (Human Machine Interface), donde serán mostrados al usuario.
- **Middleware:** Capa que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.
- **HMI (Human Machine Interface):** Es una aplicación que permite desde cualquier PC cliente acceder a los datos que se encuentran en el servidor, esta comprende todos los puntos de contacto entre el usuario y el equipo, de forma amena y evita tener que instalar software adicionales para que el sistema funcione.
- **Seguridad:** Provee las funcionalidades necesarias para garantizar el trabajo autorizado por usuarios, además brinda las herramientas para la protección contra ataques maliciosos o involuntarios al sistema por parte de personas o recursos tales como fallas de corriente, problemas de red o servidores, entre otros.
- **Reportes:** Emitir reportes o informes que consoliden la información adquirida para entregarla en un formato determinado, de tal manera que sea útil al personal que va dirigida. El funcionamiento del

subsistema encargado de la generación de reportes puede dividirse en dos: Por un lado tenemos un diseñador de informes, por otro lado existe un motor para la generación de reportes que se encarga de extraer los datos de una o varias bases de datos.

- **Configuración:** Es el encargado de almacenar, persistir y suministrar la información base para el funcionamiento de los demás módulos del SCADA. Cada módulo del SCADA, posee una biblioteca que permite establecer las comunicaciones con el servidor de configuración, a través del subsistema de comunicación, permitiendo crear, eliminar y modificar los recursos configurables del sistema. (4)

1.3.1 Las alarmas en el SCADA UX.

El SCADA UX debe permitir la representación gráfica y animada de variables de proceso y monitorización de éstas por medio de alarmas así como alertar al operador de los cambios detectados en la planta que no se consideren normales (alarmas).

Publicación de Alarmas

Las alarmas detectadas por la BDTR (Base de Datos en Tiempo Real), se mantienen en un sumario interno de alarmas activas, este sumario permite el manejo de las alarmas por parte de los operadores o el sistema de forma automática.

Cuando las alarmas son detectadas la BDTR la agrega al sumario interno de alarmas activas y las publica a todos los interesados, dentro de los que se destacan, consolas de operación, históricos, monitoreo, etc. La publicación se realiza a través de los canales de comunicación que proveen los servicios del middleware.

Una vez que la alarma es detectada y publicada, la BDTR se coloca en espera de cualquier cambio que pueda ocurrir sobre la misma, llámense comandos de operación como reconocimiento, inhibición o reinicio de la alarma o que desaparezca la causa que dio origen a la misma; si la BDTR detecta alguna de estas opciones modifica el estado del sumario interno correspondiente a esa alarma e inmediatamente publica un evento asociado a estas modificaciones y refresca a los clientes el nuevo estado de la alarma. (3)

1.4 Estándares de comunicación.

Algunos sistemas SCADA han estado sin cubrir la necesidad de comunicar de manera inmediata, o en el menor tiempo posible, los errores del sistema de producción. Es necesario combinar este elemento indispensable con algo que sirva para establecer una vía de comunicación directa cuando se produzca la

alarma en la planta. La vía de comunicación que se necesita establecer deberá de ser instantánea, barata y de fácil acceso. Son varios los sistemas SCADA que han dado solución a este problema haciendo uso de la telefonía móvil a través el envío de SMS (Servicio de mensajes cortos) del estándar GSM (Global System for Mobile Communications, Sistema global de comunicaciones móviles).

1.4.1 GSM.

La red GSM (Sistema global de comunicaciones móviles) es un estándar "de segunda generación" (2G) porque, a diferencia de la primera generación de teléfonos portátiles, las comunicaciones se producen de un modo completamente digital.

En Europa, el estándar GSM usa las bandas de frecuencia de 900MHz y 1800 MHz. Sin embargo, en los Estados Unidos se usa la banda de frecuencia de 1900 MHz. Por esa razón, los teléfonos portátiles que funcionan tanto en Europa como en los Estados Unidos se llaman tribanda y aquellos que funcionan sólo en Europa se denominan bibanda.

El estándar GSM permite un rendimiento máximo de 9,6 kbps, que permite transmisiones de voz y de datos digitales de volumen bajo, por ejemplo, mensajes de texto (SMS, Servicio de mensajes cortos) o mensajes multimedia (MMS, Servicio de mensajes multimedia). (5)

1.4.2 GPRS.

El estándar GPRS (*General Packet Radio Service*, Servicio General de Paquetes Vía Radio) es una evolución del estándar GSM y es por eso que en algunos casos se denomina GSM++ (o GMS 2+).

GPRS extiende la arquitectura del estándar GSM para permitir la transferencia de datos del paquete con una tasa de datos teóricos de alrededor de 171,2 Kbits/s (hasta 114 Kbits/s en la práctica). Gracias a su modo de transferencia en paquetes, las transmisiones de datos sólo usan la red cuando es necesario. Por lo tanto, el estándar GPRS permite que el usuario reciba facturas por volumen de datos en lugar de la duración de la conexión, lo que significa especialmente que el usuario puede permanecer conectado sin costo adicional.

GPRS admite características nuevas que no están disponibles en el estándar GSM y que se pueden clasificar en los siguientes tipos de servicios:

- Servicio de punto a punto (PTP): es la capacidad de conectarse en modo cliente-servidor a un equipo en una red IP.

- Servicio de punto a multipunto (PTMP): constituye la capacidad de enviar paquetes a un grupo de destinatarios (Multidifusión).
- Servicio de mensajes cortos (SMS). (6)

1.4.3 W-CDMA (FDD).

WCDMA (*Wideband Code Division Multiple Access* - Acceso Múltiple por División de Código de Banda Ancha) es un estándar europeo de para los sistemas inalámbricos. WCDMA ofrece flexibilidad en los servicios, combinando conmutación de paquetes y conmutación de circuitos en el mismo canal con un promedio de velocidad entre 8 Kbps hasta 2 Mbps.

La tecnología WCDMA está altamente optimizada para comunicaciones de alta calidad de voz y comunicaciones multimedia, como pueden ser las videoconferencias. También es posible acceder a diferentes servicios en un solo terminal, por ejemplo, podemos estar realizando una videoconferencia y al mismo tiempo estar haciendo una descarga de archivos muy grande, etc.

Puede soportar completamente varias conexiones simultáneas como puede ser una conexión a internet, una conversación telefónica, videoconferencia, etc. En esta plataforma se emplea estructuras de protocolos de red similares a la usada en GSM, por lo tanto está en la capacidad de utilizar redes existentes. (7)

1.4.4 TETRA.

TETRA se comenzó a crear en 1988, justo cuando se empezaban a instalar los primeros sistemas de trunking analógicos. Estas acciones hicieron que en 1991 se estableciera la tecnología TDMA como método de acceso, con cuatro canales en cada portadora de 25 KHz. La facilidad de adaptación del nuevo estándar se hacía así más viable puesto que la mayoría de sistemas PMR (*Private Mobile Radio*) empleaban un ancho de banda de 25 KHz. En 1997, se consideró que el estándar ya estaba al 100% completado y se comenzó la segunda fase instalándose los primeros sistemas. Desde entonces, el número de redes PMR basadas en TETRA son considerables y es en la actualidad una de las opciones más interesantes dentro de este mercado. (8) Este tipo de tecnología permite recibir y enviar información por medio de los radios profesionales. Gracias a este progreso, los agentes de campo se mantienen comunicados entre sí y las aplicaciones informáticas pueden interactuar con los trabajadores mediante sus radios profesionales. Su uso permite un incremento y calidad de los servicios que deben ofrecerse al usuario, con la posibilidad de alcanzar un liderazgo tecnológico, con énfasis en la eficiencia.

1.4.5 Selección del estándar.

GSM es el estándar más usado de Europa, de modo que los móviles se han convertido en una valiosa herramienta de protección y asistencia, como se pudo observar los estándares GPRS y WCDMA están basados en el estándar GSM; sin embargo, este estándar no es viable para entornos profesionales de seguridad y emergencias. En este tipo de entornos, TETRA es el sistema más adecuado pues ha sido diseñado exclusivamente para este tipo de entornos y se ha orientado desde sus inicios a satisfacer las necesidades de flotas de seguridad y emergencias con requerimientos mucho más restrictivos que los que se aplican a redes públicas. El estándar TETRA ofrece una serie de características que lo diferencian del GSM, ofreciendo una mayor privacidad y confidencialidad, más calidad de audio, mejora la velocidad de transmisión de datos, además de la capacidad de acceso a otras redes como Internet, red telefónica fija o móvil. Se orienta sobre todo hacia usuarios profesionales y corporativos que necesitan un alto grado de especialización y fiabilidad en sus comunicaciones, a un coste inferior al de la telefonía móvil GSM. Además al ser una norma abierta permite utilizar equipos de diversos fabricantes asegurando la interoperabilidad de servicios y el desarrollo de nuevas aplicaciones. Múltiples son las empresas internacionales que han potenciado el uso de este estándar, para explotar todas las ventajas que ofrece. Además los dispositivos que utilizan el estándar TETRA están diseñados para entornos industriales y preparados para soportar altas temperaturas, resistentes a choques, etc.

Para dar solución al problema que se plantea en la investigación sería muy útil la utilización de este estándar, para aprovechar sus características que lo diferencia del resto de los estándares, haciéndolo superior en algunos aspectos, y permitiendo al SCADA UX ser utilizado en empresas donde esté creada una infraestructura TETRA. Los usuarios profesionales como son los sectores de la seguridad pública y el transporte son los que han generado mayor demanda del mismo, debido a la robustez que muestra el estándar en lo que tiene que ver con seguridad en la comunicación (encriptamiento en la interfaz aire y libertad para que los fabricantes implementen seguridad en los terminales) y también la posibilidad de conectarse con otras redes. El CEDIN cuenta actualmente con los dispositivos TETRA, lo que hace posible el desarrollo de la solución a partir del estándar TETRA.

1.5 Estándar TETRA.

TETRA es un estándar elaborado por el ETSI (Instituto Europeo de Estándares de Telecomunicación) que ha reunido propuestas de operadores de redes, administraciones nacionales, fabricantes de equipos y

usuarios de servicios móviles para establecer una norma abierta para las comunicaciones móviles digitales profesionales, o sea, define un sistema móvil digital de radio. (9)

Está enfocado fundamentalmente a sectores críticos como son los servicios de seguridad y emergencias (policía, bomberos, ambulancias…) para servicios de voz y datos. Los servicios avanzados de transmisión de datos presentan multitud de aplicaciones de valor añadido, como por ejemplo: telecontrol, consulta de bases de datos, posicionamiento y gestión de flotas, sistemas de telepago, transmisión de archivos, etc. Esto permite crear una gran gama personalizada de productos adaptada a cada uno de los usuarios de servicios TETRA.

1.5.1 Características.

Estándar abierto – respaldado por múltiples fabricantes independientes de infraestructura y terminales lo cual ofrece gran cantidad de opciones y seguridad a los usuarios.

Versatilidad – TETRA cuenta con muchas características de los sistemas PMR y PAMR (PAMR, *Public Access Mobile Radio* conocido también como *Private Access Mobile Radio*). Por ejemplo, proporciona llamadas de grupo, de difusión y de emergencia, puesta en cola de llamadas, prioridades y modo directo. Además, también ofrece VPN (*Virtual Private Network)* proporcionando un sistema compartido en que los servicios de emergencia tienen comunicaciones privadas pero pueden hablar juntos cuando estén atendiendo un incidente mayor.

Tiempo de establecimiento de llamada rápido – menor a los 0.3s.

Seguridad – Puede implementar tres niveles de seguridad: autenticación, encriptación en el aire y encriptación extremo a extremo.

Capacidad de transmisión de Datos – Las redes TETRA ofrecen servicios de datos desde mensajes de estado o mensajes cortos hasta servicios de datos para acceder a bases de datos remotas, etc. Por tanto, aunque TETRA no disponga de una gran velocidad de transmisión el hecho de que pueda ofrecer múltiples servicios es una gran ventaja.

Interoperabilidad de terminales – El sistema permite disponer de terminales de diferentes proveedores operando dentro de una única red (que, a su vez, puede ser de otro proveedor diferente) y comunicándose entre sí de manera totalmente compatible. (8)

1.5.2 Servicios que ofrece.

El desarrollo de la norma TETRA se ha orientado a potenciar los servicios que se requerirán en las modernas aplicaciones de PAMR. Se aprovecha además la flexibilidad que ofrece la técnica TDMA para proporcionar transmisiones simultáneas de voz y datos. Se ofrecen diferentes velocidades de datos, de hasta 28.8 Kbps para servicios de datos en modo circuito, dependiendo de los canales asignados a la comunicación y el grado de protección de los datos. De acuerdo a esto se categoriza a los servicios en: Teleservicios o servicios de voz, servicios de portadora o servicios de datos y servicios adicionales o suplementarios.

Los teleservicios permiten la comunicación entre usuarios y se realizan mediante acuerdos establecidos entre los diferentes operadores de red, en este acuerdo se incluyen las funciones de los quipos terminales. Hacen uso de las capas superiores del modelo OSI. Entre estos se encuentran: Llamada individual, llamada grupal, llamada de difusión, llamada de emergencia, operación en modo directo DMO, canal abierto, inclusión de llamada.

Los servicios portadores permiten la transmisión de señales entre diferentes puntos de acceso. Se realizan hasta la capa Red del modelo OSI. Entre estos se encuentran: transmisión del estado de usuario, servicios de datos cortos, servicio de datos sobre circuitos conmutados, servicio de datos sobre paquetes conmutados.

Los servicios suplementarios son los que modifican o complementan un servicio portador o un teleservicio. Entre estos los servicios esenciales son: llamada autorizada por despachador, acceso prioritario, prioridad de llamada, acceso tardío, escucha discreta, ambiente de escucha. (10)

1.5.3 Servicios de datos.

Existen cuatro tipos de comunicaciones básicas de datos en el sistema TETRA: Mensajes de estado, mensajes cortos, datos en modo circuito y datos en modo paquete.

Mensajes de estado: Este servicio permite transmitir y recibir mensajes pre-codificados a través de la red TETRA.

Datos en modo paquete: Este servicio permite extender las comunicaciones de datos para actuar como una sub-red IP.

Datos en modo circuitos: Este servicio es el más potente en términos de transferencia de datos, donde el MS (Mobile Station) actúa como un módem transparente sobre la red TETRA.

Servicio de datos cortos (SDS): El Servicio de datos cortos (SDS) permite transmitir y recibir mensajes cortos pre-definidos o definidos por el usuario a través de la red TETRA.

- Soporta transmisiones de datos punto a punto o punto-multipunto.Según el tamaño del mensaje se distinguen varios tipos de SDS:
  - •Tipo 1 (2 bytes) , tipo 2 (4 bytes), tipo 3 (8 bytes).
  - •Tipo 4 (longitud variable):
    - ✓ 140 bytes (formato ISO 8-bit) / 160 bytes (formato GSM-7bit) cuando se emplea “LLC Basic Link”.
    - ✓ 256 bytes usando “LLC AdvancedLink”(no cubierto por los perfiles de interoperabilidad).
- Dependiendo del tamaño de los SDS, se podrán emplear mecanismos de stealing para enviar o recibir mensajes durante llamadas en modo circuito (voz, datos).
- SDS tiene un tiempo de respuesta rápido (del orden de milisegundos).
- El SDS-TransportLayer(TL) es un protocolo adicional (opcional) para SDS tipo 4 que mejora el servicio por medio de reconocimiento extremo a extremo, almacenamiento y reenvío (con centro de servicios en la SwMI) y soporte para múltiples protocolos de aplicaciones.
- El uso de diferentes Identificadores de Protocolo (PIDs) previene a las aplicaciones de interacciones o conflictos no deseados. La asignación de PID corresponde al ETSI quien centraliza esta información para asegurar futura interoperabilidad y evitar congestión de identificadores. (11)

1.6 Conclusiones parciales.

En este capítulo se trataron diferentes temas, como los sistemas SCADA centrándose fundamentalmente el estudio en el SCADA UX y en la notificación de las alarmas en dicho sistemas, así como en la selección del estándar a utilizar en la solución y el servicio de envío de mensajes del mismo, fundamentales en la solución al problema planteado.

CAPÍTULO 2: TECNOLOGÍAS Y HERRAMIENTAS

2.1 Introducción.

Actualmente existen disímiles herramientas y tecnologías para dar soluciones a problemas complejos que se presentan a diario y que favorece un mayor desempeño de grandes y medianas empresas. En el siguiente capítulo se argumentará sobre las tecnologías y herramientas utilizadas en el desarrollo de software, que en este caso no son objeto de selección, pues las mismas fueron analizadas y evaluadas desde los inicios del proyecto SCADA.

2.2 Sistema Operativo. Distribución de Linux: Debian GNU/Linux.

Debian Linux es una distribución de Linux completamente nueva. En vez de estar desarrollada por un individuo aislado o un grupo, se desarrolla abiertamente en el espíritu de Linux y GNU. Comenzó como un grupo de pocos y fuertemente unidos hackers de Software Libre; es también un intento por crear una distribución no comercial que será capaz de competir efectivamente en el mercado comercial. Algunas de las características más importantes del mismo se describen a continuación:

- Coste: Debian es un sistema operativo (S.O.) de libre distribución (es decir sin coste alguno).
- Multiusuario: permite a varios usuarios acceder al mismo tiempo a través de terminales, y distribuye los recursos disponibles entre todos.
- Multiplataforma: Es decir que puede correr en la mayoría de plataformas del mercado (procesadores de la gama Intel y AMD, Motorola, Sun, Sparc, etc.).
- Licencia: Debian nace como una apuesta por separar en sus versiones el software libre del software no libre, para esto debe respetar 4 libertades: 1.libertad para usarlo. 2. libertad para modificarlo. 3. libertad para copiarlo. 4. libertad para distribuir las modificaciones.
- Seguridad: los problemas de seguridad se solucionan rápidamente con parches de seguridad que se actualizan en internet.
- Facilidad de Uso: este sistema operativo no es aconsejable para aquellas personas procedentes de Windows, que quieren instalarse Linux porque han oído hablar muy bien de él, pero que quieren una instalación lo más parecida a Windows. (12)

2.3 Metodología de desarrollo del software. RUP

Se entiende por metodología de desarrollo una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software. La finalidad de una metodología de desarrollo es garantizar la eficacia y la eficiencia en el proceso de generación de software.

RUP

RUP (*Rational Unified Process*) es un Proceso Unificado propuesto por IBM actualmente considerado como un estándar en el desarrollo de software en las empresas. Es un modelo que permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación, garantizando el cumplimiento de ciertos estándares de calidad. RUP define claramente quien, cómo, cuándo y qué debe hacerse en un proyecto. Posee tres características esenciales:

- Está dirigido por los Casos de Uso: que orientan el proyecto a la importancia para el usuario y lo que este quiere.
- Está centrado en la arquitectura: que relaciona la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden.
- Es iterativo e incremental: donde divide el proyecto en mini proyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada.

RUP divide el proceso en 4 fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. Inicio: Se hace un plan de fases, se identifican los principales casos de uso y se identifican los riesgos. Se define el alcance del proyecto. Elaboración: se hace un plan de proyecto, se completan los casos de uso y se eliminan los riesgos. Construcción: se concentra en la elaboración de un producto totalmente operativo y eficiente y el manual de usuario. Transición: se instala el producto en el cliente y se entrena a los usuarios. Las disciplinas de RUP están claramente relacionadas a las 6 mejores Prácticas, pero más de cerca representa roles de los miembros o subgrupos dentro del equipo de desarrollo completo. Estas disciplinas son:

1. Modelado del negocio: Entendiendo las necesidades del negocio
2. Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.
3. Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.
4. Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

5. Prueba: Asegurándose que el comportamiento requerido es el correcto y que todo lo solicitado está presente.
6. Despliegue: Hacer todo lo necesario para la salida del proyecto.
7. Manejo de Proyecto: Administrando horarios y recursos.
8. Ambiente: Administrando el ambiente de desarrollo.
9. Configuración y Manejo del Cambio: Guardando todas las versiones del proyecto.

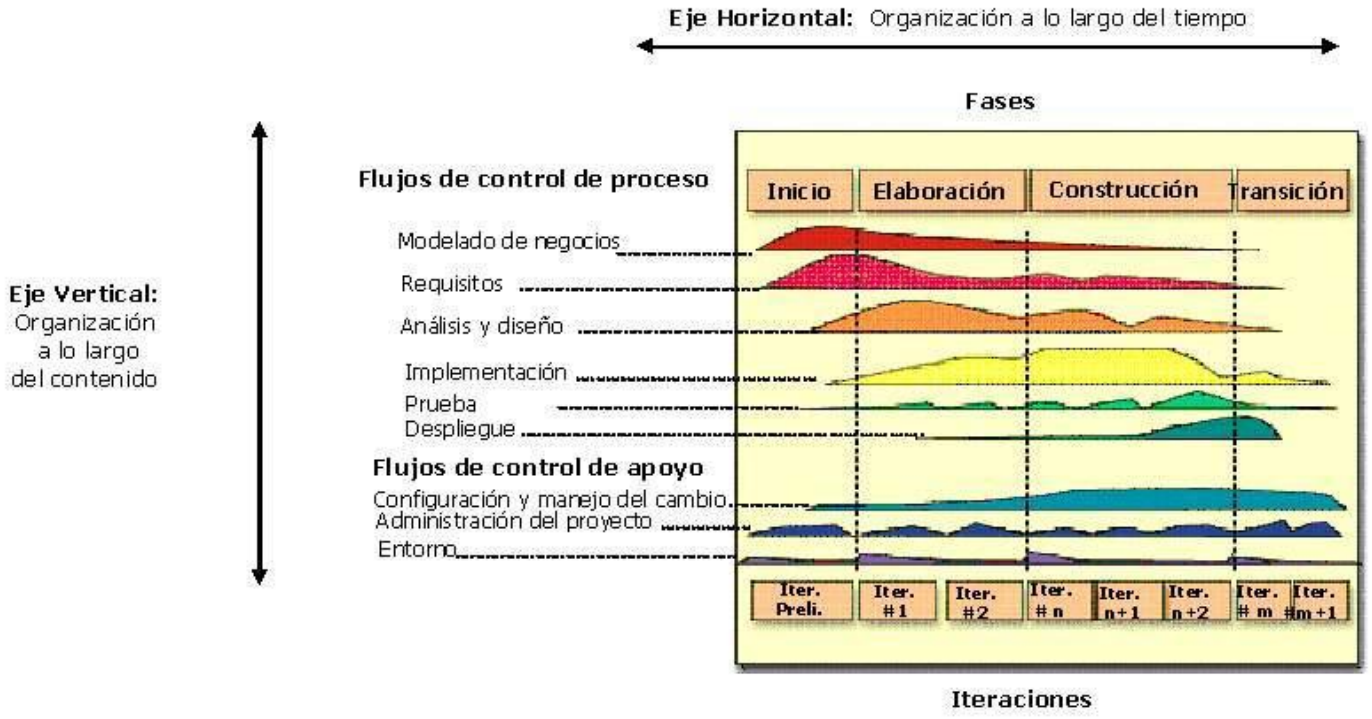

Fig. 2 Estructura de RUP.

Al utilizar una metodología de desarrollo como RUP estamos asegurando que nuestro proceso se realice de manera eficiente. Esta metodología de desarrollo permite que el equipo aproveche el tiempo, permitiendo que el software sea realizado iterativamente, esto trae consigo que el cliente sea parte activa en este proceso y que se detecten de manera temprana los errores. Permite además que se haga una buena administración de los requerimientos, donde se muestra como los escenarios utilizadas en RUP han demostrado ser una manera excelente de capturar los requerimientos funcionales y asegurarse que direccionan el diseño, la implementación y la prueba del sistema, logrando así que el sistema satisfaga las necesidades del usuario. Cumple con otras características muy importantes como verificar la calidad del

software, el aseguramiento de la calidad se construye dentro del proceso, en todas las actividades, involucrando a todos los participantes, utilizando medidas y criterios objetivos, permitiendo así detectar e identificar los defectos en forma temprana y controlar los cambios en el software donde la capacidad de administrarlos es esencial en ambientes en los cuales el cambio es inevitable. RUP describe como controlar, rastrear y monitorear los cambios para permitir un desarrollo iterativo exitoso.

2.4 Lenguaje unificado de modelado. UML

UML (*Unified Modeling Language)* o lo que es lo mismo Lenguaje Unificado de Modelado es, precisamente, un lenguaje de modelado que permite la representación conceptual y física de un sistema. Las características principales de este lenguaje son: Aporta una notación estándar orientada a objetos. Permite describir un sistema en diferentes niveles de abstracción. Divide cada proyecto en un número de diagramas que representan diferentes vistas del proyecto. UML se puede aplicar tanto a sistemas informáticos como a sistemas que no son informáticos.

Con la utilización de UML es posible llevar a cabo las siguientes funciones:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión. (13)

2.5 Herramienta case. Visual Paradigm.

Las herramientas CASE (*Computer Aided Software Engineering*), en español Ingeniería de Software Asistida por Computadora, son aplicaciones informáticas utilizadas en el proceso de desarrollo de software en tareas como: realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente a partir del diseño, compilación automática, documentación o detección de errores entre otras.

Visual Paradigm es una herramienta 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. Las características principales de esta herramienta son las que a continuación se enuncian:

- 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. (14)

2.6 Lenguaje de programación. C++

El lenguaje de programación C++ fue creado en los años 80 por Bjarne Stroustrup basando en el lenguaje C. Es un lenguaje orientado a objetos al que se le añadieron características y cualidades de las que carecía el lenguaje C; surge con el objetivo de añadir a C nuevas características: clases y funciones, tipos genéricos, expresiones así como la posibilidad de declarar variables en cualquier punto del programa. Es multiplataforma, o sea se puede usar en diferentes sistemas operativos. Este lenguaje depende mucho del hardware, es uno de los más potentes porque nos permite programar a alto y a bajo nivel, es complicado porque debemos hacerlo nosotros mismos casi todo.

Algunas de las características que lo distinguen de los demás lenguajes de programación son:

- Programación orientada a objetos: La posibilidad de orientar la programación a objetos permite al programador diseñar aplicaciones desde un punto de vista más cercano a la vida real. Además, permite la reutilización del código de una manera más lógica y productiva.
- Portabilidad: Un código escrito en C++ puede ser compilado en casi todo tipo de ordenadores y sistemas operativos sin hacer apenas cambios.
- Brevedad: El código escrito en C++ es muy corto en comparación con otros lenguajes, sobre todo porque en este lenguaje es preferible el uso de caracteres especiales que las "palabras clave".
- Programación modular: Un cuerpo de aplicación en C++ puede estar hecho con varios ficheros de código fuente que son compilados por separado y después unidos. Además, esta característica

permite unir código en C++ con código producido en otros lenguajes de programación como Ensamblador o el propio C.

- Velocidad: El código resultante de una compilación en C++ es muy eficiente, gracias a su capacidad de actuar como lenguaje de alto y bajo nivel y a la reducida medida del lenguaje. (15)

2.7 Entorno integrado de desarrollo (IDE). Eclipse.

Eclipse es un Entorno Integrado de Desarrollo, del inglés *Integrated Development Environment* (IDE), para todo tipo de aplicaciones libres, inicialmente desarrollado por IBM, y actualmente gestionado por la Fundación Eclipse.

Herramienta para el programador. Esta herramienta fue desarrollada principalmente para el desarrollo de aplicaciones Java, facilitando al máximo la gestión de proyectos colaborativos mediante el control de versiones 'cvs', es posible también con subversión, exportar e importar proyectos.

Es posible añadir nuevas funcionalidades al editor, a través de nuevos módulos ('plugins'), para programar en otros lenguajes de programación además de Java como C/C++, PHP, Python, Ruby y Cobol.

Características actuales.

- Multiplataforma (GNU/Linux, Solaris, Mac OSX, Windows).
- Soportado para distintas arquitecturas (x86, 64).
- Estructura de plug-in que hace sencillo añadir nuevas características y funcionalidades.
- Control de versiones con cvs o con subversión. Resaltado de sintaxis, autocompletado, tabulador de un bloque de código seleccionado, es decir varias utilidades de edición que ayudan enormemente al programador. (16)

2.8 Conclusiones parciales.

Después de la explicación de las herramientas y tecnología a utilizar, este capítulo sirve de base para el desarrollo de la propuesta de solución al problema planteado.

CAPÍTULO 3: PRESENTACIÓN Y DISEÑO DE LA SOLUCIÓN

3.1 Introducción.

En este capítulo se aborda todo el proceso desde la captura de los requisitos, tantos funcionales, como no funcionales hasta el diseño de la solución que se propone, el cual va a ser representado mediante artefactos ingenieriles, tales como los diagramas de clases de diseño, de despliegue, y los de caso de usos del sistema, así como también quedará reflejada la arquitectura del software, y los patrones del diseño, todo esto con vista a darle solución al problema planteado.

3.2 Propuesta de solución.

El sistema que se propone es un módulo que tiene como objetivo de notificar la ocurrencia cualquier alarma que llegue al sistema como mensaje de texto a través de los radios TETRA. La aplicación le permitirá al emisor realizar la solicitud para el envío del mensaje, y recibirá una notificación después de enviado el mismo.

El sistema está basado en la arquitectura cliente-servidor, donde la aplicación cliente recibirá la solicitud por parte del emisor para enviar el mensaje, y con dicha solicitud el cuerpo mensaje y el número del radio al que se enviará el mismo; y la aplicación servidor estará conectada al dispositivo, en este caso, el radio TETRA, quien finalmente será el encargado de enviar el mensaje a otro radio. El contenido del mensaje estará relacionado con las alarmas que se desee enviar, puede ser la misma alarma o una notificación de la existencia de la misma; estas son procesadas por la aplicación cliente quien le entrega a la aplicación servidor el mensaje listo para enviar.

El mensaje a enviar puede proceder de distintos remitentes, por ejemplo en caso de que la notificación se envíe automáticamente, el SCADA debe incluir una función que se encargue de recibir las alarmas provenientes de la BDTR y extraiga la información a enviar, y lo asocie al número al que se debe enviar, y luego entregar esta información a la aplicación cliente; en caso de que sea el operador quien envíe, habría que incluirle al HMI una interfaz visual que permita introducir el contenido del mensaje y el número asociado a dicho mensaje, esta operación debe realizarse inmediatamente cuando llegue la alarma al HMI.

3.3 Especificación de los Requisitos del software.

A partir de este punto se modela la solución propuesta. Para ello se identifican sus requisitos, tanto funcionales como no funcionales, y se modelan las funcionalidades en términos de casos de uso del sistema.

3.3.1 Requisitos Funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir.

El sistema debe permitir:

RF 1. Recibir la información del mensaje a enviar.

RF 2. Enviar el mensaje de texto.

- **RF 2.1** Conectarse con el dispositivo y configurarlo.
- **RF 2.2** Enviar la información recibida como mensaje de texto.

3.3.2 Requisitos No Funcionales.

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, puesto que las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable es el sistema, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación. Existen varios tipos de requisitos no funcionales. A continuación se presentan los correspondientes a la solución propuesta.

RNF 1. Software

✓ **RNF 1.1** La notificación de alarmas por mensaje de texto se debe desarrollar sobre el Sistema operativo GNU/Linux, distribución Debian, Kernel 2.6

RNF 2. Diseño e Implementación

✓ **RNF 2.1** Para el desarrollo de la notificación de alarmas por mensaje de texto se debe utilizar el lenguaje de programación C++.

✓ **RNF 2.2** Para el desarrollo de la notificación de alarmas por mensaje de texto se debe utilizar como paradigma de programación, la Programación Orientada a Objetos.

✓ **RNF 2.3** Para el análisis y diseño de la aplicación se utiliza Visual Paradigm como herramienta Case.

RNF 3. Portabilidad

✓ **RNF 3.1** La notificación de alarmas por mensaje de texto debe garantizar su funcionalidad sin

importar la plataforma que se utilice.

RNF 4. Fiabilidad

✓ **RNF 4.1** La notificación de alarmas por mensaje de texto debe garantizar una comunicación eficiente y fiable para evitar las pérdidas de información.

✓ **RNF 4.2** La solución debe brindar garantía de un tratamiento adecuado de las excepciones.

RNF 5. Soporte

✓ **RNF 5.1** La notificación de alarmas por mensaje de texto debe permitir una amplia interoperabilidad para que aplicaciones elaboradas por terceros e implementadas en diferentes lenguajes y sistemas operativos puedan comunicarse con este de forma fácil y eficiente.

3.4 Arquitectura de software.

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta. Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. La interacción cliente-servidor es el soporte de la mayor parte de la comunicación por redes. Ayuda a comprender las bases sobre las que están construidos los algoritmos distribuidos.

El servidor debe negociar con su Sistema Operativo un puerto (casi siempre bien conocido) donde esperar las solicitudes. El servidor espera pasivamente las peticiones en un puerto bien conocido que ha sido reservado para el servicio que ofrece. El cliente también solicita, a su sistema operativo, un puerto no usado desde el cual enviar su solicitud y esperar respuesta. Un cliente ubica un puerto arbitrario, no utilizado y no reservado, para su comunicación.

En una interacción se necesita reservar solo uno de los dos puertos, asignados un identificador único de puerto para cada servicio, se facilita la construcción de clientes y servidores.

Los servidores por lo general son más difíciles de construir que los clientes pues aunque se implantan como programas de aplicación deben manejar peticiones concurrentes, así como reforzar todos los procedimientos de acceso y protección del sistema computacional en el que corren, y protegerse contra todos los errores posibles. El cliente y el servidor pueden interactuar en la misma máquina. (17)

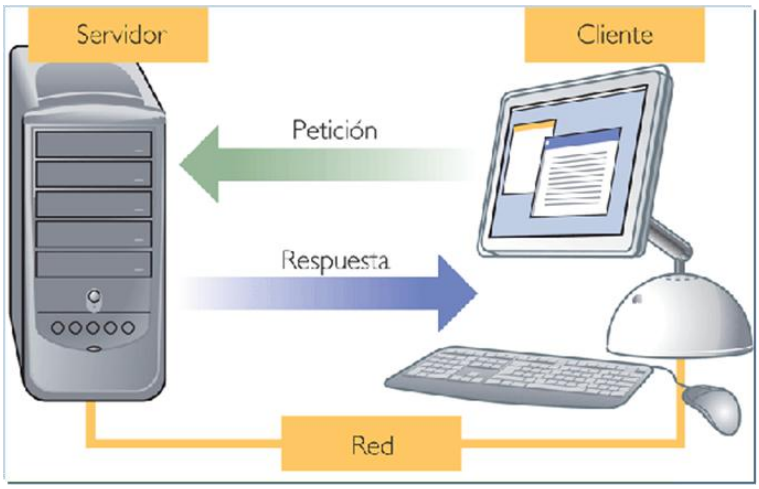

Fig. 3 Arquitectura Cliente-Servidor.

3.4.1 Características.

- Combinación de un cliente que interactúa con el usuario, y un servidor que interactúa con los recursos a compartir. El proceso del cliente proporciona la interfaz entre el usuario y el resto del sistema. El proceso del servidor actúa como un motor de software que maneja recursos compartidos tales como bases de datos, impresoras, Módem, etc.
- Las tareas del cliente y del servidor tienen diferentes requerimientos en cuanto a recursos de cómputo como velocidad del procesador, memoria, velocidad y capacidades del disco e input-output devices.
- Se establece una relación entre procesos distintos, los cuales pueden ser ejecutados en la misma máquina o en máquinas diferentes distribuidas a lo largo de la red.
- Existe una clara distinción de funciones basadas en el concepto de “servicio”, que se establece entre clientes y servidores.
- La relación establecida puede ser de muchos a uno, en la que un servidor puede dar servicio a muchos clientes, regulando su acceso a los recursos compartidos.
- Los clientes corresponden a procesos activos en cuanto a que son estos los que hacen peticiones de servicios. Estos últimos tienen un carácter pasivo, ya que esperan peticiones de los clientes.
- No existe otra relación entre clientes y servidores que no sea la que se establece a través del intercambio de mensajes entre ambos. El mensaje es el mecanismo para la petición y entrega de solicitudes de servicios.

- El ambiente es heterogéneo. La plataforma de hardware y el sistema operativo del cliente y del servidor no son siempre los mismos. Precisamente una de las principales ventajas de esta arquitectura es la posibilidad de conectar clientes y servidores independientemente de sus plataformas.
- El concepto de escalabilidad tanto horizontal como vertical es aplicable a cualquier sistema Cliente-Servidor. La escalabilidad horizontal permite agregar más estaciones de trabajo activas sin afectar significativamente el rendimiento. La escalabilidad vertical permite mejorar las características del servidor o agregar múltiples servidores. (17)

3.5 Modelo de casos de uso del sistema.

En esta sección se identifican los actores del sistema a desarrollar, y se definen los casos de uso del sistema, así como la descripción textual de los mismos.

3.5.1 Actores del sistema.

Los actores de un sistema son agentes externos, roles que las personas (usuarios) o dispositivos juegan cuando interactúan con el software.

Actores	Descripción
Emisor	Subsistema u operador del sistema que hace la solicitud de enviar el mensaje.

Tabla 1. Descripción de los actores del sistema.

3.5.2 Casos de uso del sistema.

Los casos de uso especifican una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia.

El caso de uso de este sistema es: Gestionar envío de mensaje.

3.5.3 Diagrama de casos de uso del sistema.

Los diagramas de casos de uso se crean para visualizar las relaciones entre los actores y los casos de uso.

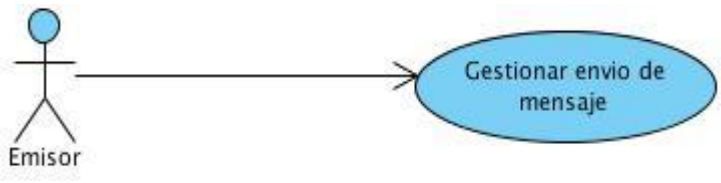

Fig. 4 Diagrama de casos de uso del sistema.

3.5.4 Descripción del los casos de uso del sistema.

Cada caso de uso tiene una descripción de las funcionalidades que ejecutará el sistema propuesto como respuesta a las acciones del usuario. Las tablas presentadas a continuación argumentan los flujos operacionales de cada uno de ellos.

Caso de uso	**Gestionar envío de mensaje**
Actores	Emisor
Propósito	Enviar la alarma recibida como mensaje de texto.
Resumen	El caso de uso se inicia cuando la aplicación cliente recibe una petición del usuario para enviar una alarma como mensaje de texto.
Precondiciones	El SCADA ha sido instalado. La información del mensaje fue recibida correctamente.
Referencias	RF1,RF2, RF2.1, RF2.2
Prioridad	Crítico.
Flujo Normal de los eventos.	
Acción del actor.	**Respuesta del sistema.**
1. Manda la petición para enviar el mensaje de texto.	1.1 La aplicación cliente recibe dicha petición con la información así como el número al que se va enviar la misma. 1.2 La aplicación cliente se conecta a la aplicación servidor y le envía la información que quiere enviar como mensaje de texto, y el número al que lo va a enviar. 1.3 La aplicación servidor recibe esta información y se conecta con el dispositivo que va a enviar el mensaje, en este caso, el radio

1.8 Recibe la confirmación del envío del mensaje. 1.9 Termina el caso de uso.	TETRA. 1.4 La aplicación servidor configura el radio y le envía el mensaje. 1.5 El radio envía el mensaje. 1.6 La aplicación servidor envía a la aplicación cliente una confirmación del envío del mensaje. 1.7 La aplicación cliente envía al emisor la confirmación de envío del mensaje.
Poscondiciones	El mensaje se envía a los operadores que tengan los otros radios.

Tabla 2. Descripción del caso de uso Gestionar envío de mensaje.

3.6 Diseño del sistema.

Según Presman, el diseño es una representación significativa de ingeniería de algo que se va a construir. Se puede hacer el seguimiento basándose en los requisitos del cliente, y al mismo tiempo la calidad se puede evaluar y cotejar con el conjunto de criterios predefinidos para obtener un buen diseño.

El diseño del software es un proceso iterativo mediante el cual los requisitos se traducen en un plano para construir el software y es tanto un proceso como un modelo, es una secuencia de pasos que hacen posible al diseñador describir como se va a construir el software. Durante el diseño orientado a objetos, se presta especial atención a la definición de los objetos software y en cómo colaboran para satisfacer los requisitos. (18)

3.6.1 Patrones del diseño.

Los patrones de diseño son “Soluciones ya probadas y eficaces de los problemas de diseño que pueden expresarse como un conjunto de principios”. Una de las principales razones de las ciencias de la computación en cuanto a uso de los patrones de diseño, es la necesidad de obtener soluciones elegantes, simples y reutilizables. (18) Con el objetivo de alcanzar mayor calidad en el diseño se tuvo en cuenta los siguientes patrones de diseño:

Experto: Indica el principio básico de asignación de responsabilidad, usado durante la implementación del sistema para asignar responsabilidades a las clases que cuentan con la información necesaria para

cumplirlas. Su uso proporciona un sistema más fácil de mantener y ampliar, además ofrece la posibilidad de reutilizar los componentes en futuras aplicaciones. Este patrón está presente en la interface ISDSTETRAService, que se entrega para que cada servidor la implemente a su manera, y a la hora de invocar un método, responde como de implemento en cada servidor.

Simple Factory: es un patrón de creación que se encarga de crear instancias de objetos de manera que ya no se tendrán que instanciar directamente proporcionando a los programas una mayor flexibilidad para decidir qué objetos usar. Este patrón está presente en las clases SDSTETRAClientResolver y SDSTETRAServerResolver, pues estas clases son fábricas para crear y devolver el objeto que se define en su implementación, en estos casos crearía SDSTETRAClient, y SDSTETRAServer, respectivamente, implementados con ice.

Strategy: es un patrón de Comportamiento que permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución. Este patrón está presente en la interface que se brinda ISDSTETRAService, donde en este momento un método solo tiene una implementación y a la hora de responder, responde de la manera que le implementaron.

3.6.2 Modelo del diseño.

El modelo de diseño es el conjunto de diagramas que describen el diseño lógico. Su propósito es lograr una abstracción de la implementación del sistema. Es usado para concebir un documento del diseño del sistema de software. Es abarcador, compuesto por artefactos que engloban todas las clases del diseño, subsistemas, paquetes, colaboraciones, y las relaciones entre ellos.

3.6.2.1 Diagrama de Secuencia.

Un diagrama de secuencia es una forma de diagrama de interacción que muestra los objetos como líneas de vida a lo largo de la página y con sus interacciones en el tiempo representadas como mensajes dibujados como flechas desde la línea de vida origen hasta la línea de vida destino. Los diagramas de secuencia son buenos para mostrar qué objetos se comunican con qué otros objetos y qué mensajes disparan esas comunicaciones. Los diagramas de secuencia no están pensados para mostrar lógicas de procedimientos complejos.

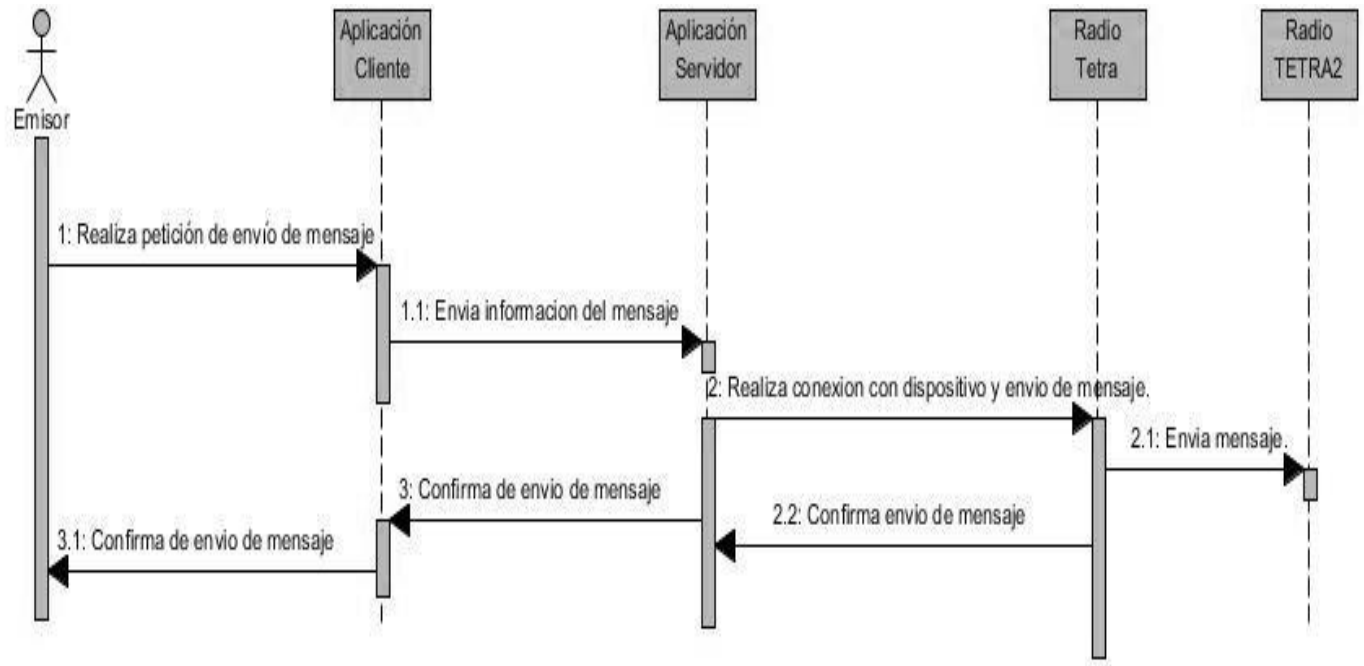

Fig. 5 Diagrama de secuencia.

3.6.2.2 Paquetes del diseño.

Los paquetes son una colección de clases, relaciones, realizaciones de casos de usos, diagramas y otros paquetes que son empleados para dividir en partes más pequeñas al modelo de diseño. Estos son utilizados para agrupar un conjunto cohesivo de responsabilidades, elementos del modelo de diseño que se relacionen y como herramienta organizacional.

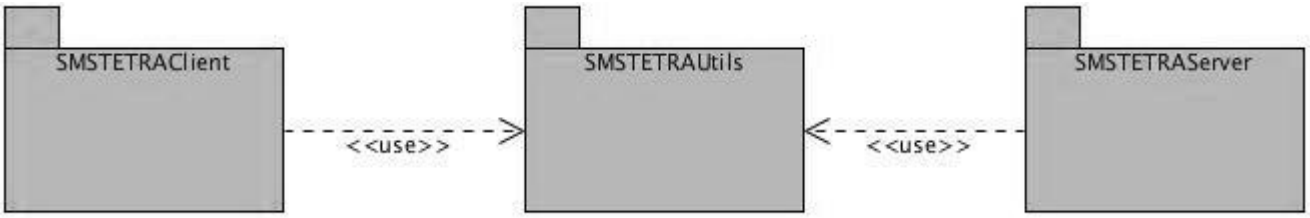

Fig. 6 Diagrama de paquetes.

SDSTETRAUtils: Este paquete contiene las definiciones de ICE compiladas con el compilador de ice que genera código c++, así como las clases y sus relaciones correspondientes al mismo.

SDSTETRAClient: Contiene todas las clases y sus relaciones correspondientes a la aplicación por parte del cliente.

SDSTETRAServer: Contiene todas las clases y sus relaciones correspondientes a la aplicación por parte del servidor.

3.6.2.3 Diagrama de clases del diseño.

Los diagramas de clases del diseño (DCD) representan las especificaciones de las clases e interfaces software en una aplicación.

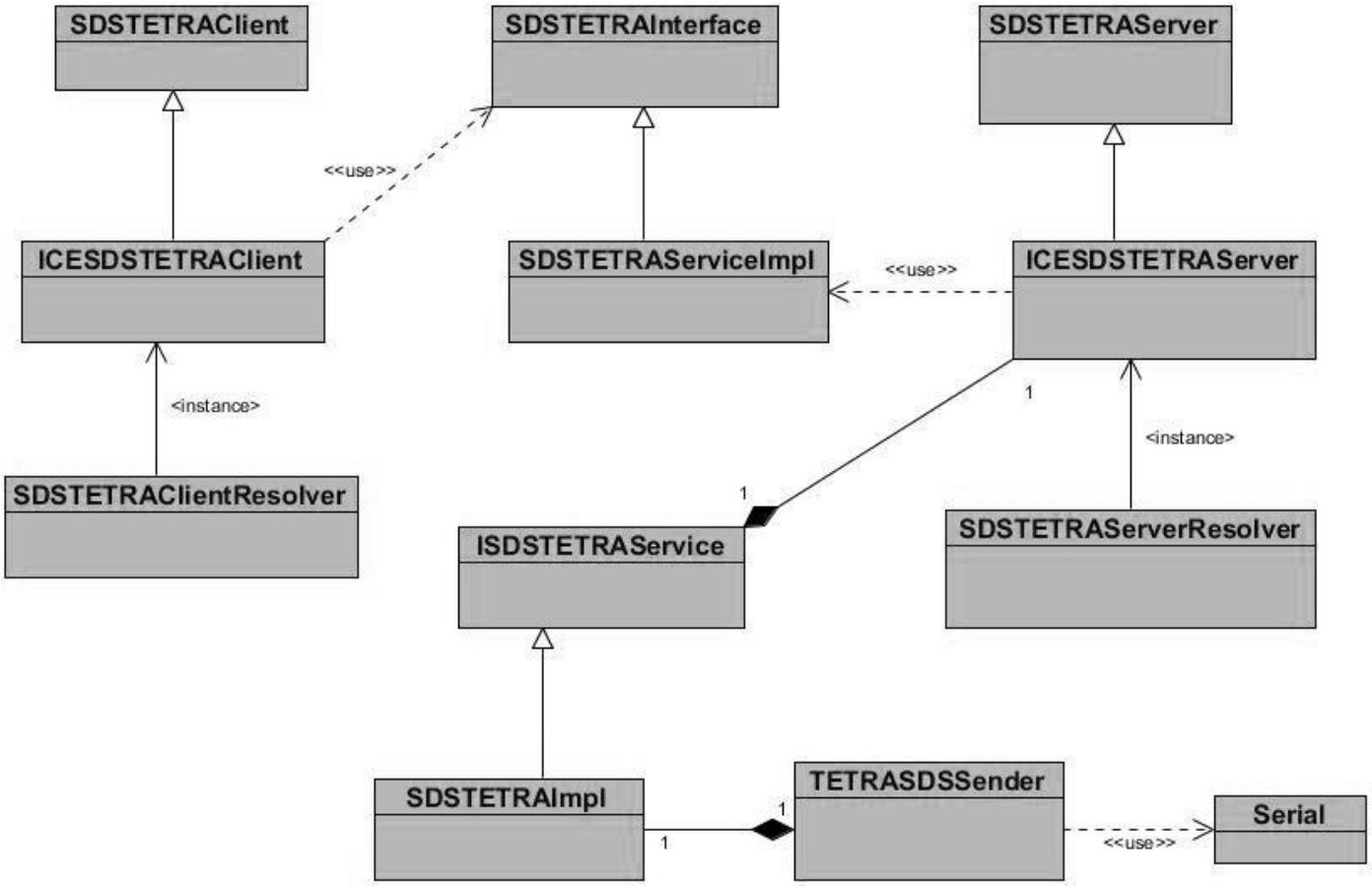

Fig. 7 Diagrama de clases del diseño.

Descripción de las clases

SDSTETRAServer: Clase servidor que maneja el envío de mensajes de texto para el SCADA UX.

ICESDSTETRAServer: En esta clase se realiza la implementación del servidor que envía mensajes de texto del SCADA UX basado en Ice de Zeroc.

SDSTETRAServerResolver: Fábrica de Servidores para el envío de mensajes de texto del SCADA UX.
SDSTETRAClient: Cliente de envío de mensajes de texto para el SCADA UX.
ICESDSTETRAClient: En esta clase se realiza la implementación del cliente que enviará el mensaje de texto para la configuración del SCADA UX. Esta implementación es utilizando la tecnología Ice de Zeroc.
SDSTETRAClientResolver: Fábrica para crear clientes de mensajes del SCADA UX.
SDSTETRAServiceImpl: En esta clase se realiza la implementación de la interfaz de envío de mensajes de texto.
ISDSTETRAService: Es una interfaz que define los métodos que luego se van a implementar.
SDSTETRAInterface: En esta clase se encuentra todo lo necesario para crear una conexión usando ice.
TETRASDSSender: En esta clase se gestiona el envío de los SDS.
Serial: En esta clase se maneja el puerto de serie, o sea, se realizan todas las operaciones necesarias sobre el mismo.

3.7 Conclusiones parciales.

En este capítulo se mostró la propuesta del sistema, así como los requerimientos del software los cuales se deben cumplir para lograr el resultado esperado. También se realizó una descripción detallada de los casos de uso, los cuales son representados en un diagrama de casos de uso del sistema. Se mostró el modelo del diseño, incluyéndose diagramas de clases para lograr una mejor representación de estas en el sistema. Estos modelos sirvieron de ayuda para lograr una comprensión detallada de los requerimientos del software. Además se realizó un análisis de los patrones de arquitectura y diseño, contribuyendo a obtener la calidad requerida, y de esta forma dar paso a la implementación del sistema.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS DE LA SOLUCIÓN

4.1 Introducción.

Este capítulo comienza con el resultado del diseño, implementando el sistema en términos de componentes. Además se hace una revisión final de las especificaciones del diseño y de la codificación mediante la realización de pruebas que garanticen el correcto funcionamiento del software.

4.2 Implementación.

El flujo de trabajo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

Los diagramas de despliegue y componentes, que son artefactos generados en este flujo de trabajo conforman lo que se conoce como un modelo de implementación al describir los componentes a construir y su organización y dependencia entre nodos físicos en los que funcionará la aplicación.

4.2.1 Diagrama de componentes.

Los diagramas de componentes representan un conjunto de componentes que se relacionan entre sí mediante dependencias, estos pueden ser ejecutables, librerías, ficheros de código, una tabla de base de datos, un documento etc.

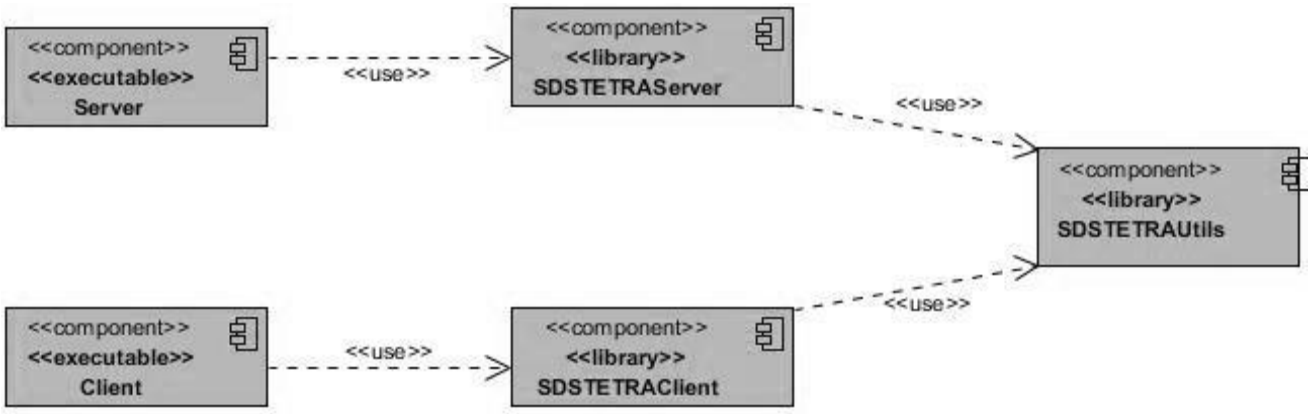

Fig. 8 Diagrama de componentes.

Descripción de los componentes.

Server: Ejecutable que se genera para la aplicación servidor.

Client: Ejecutable que se genera para la aplicación cliente.

SDSTETRAServer: Biblioteca que contiene la implementación relacionada con el servidor.

SDSTETRAClient: Biblioteca que contiene la implementación relacionada con el cliente.

SDSTETRAUtils: Biblioteca que encapsula las definiciones de ice con los códigos generados al compilar el .ice. Define los métodos que el cliente encuesta al servidor.

4.2.1 Diagrama de despliegue.

Los diagramas de despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación.

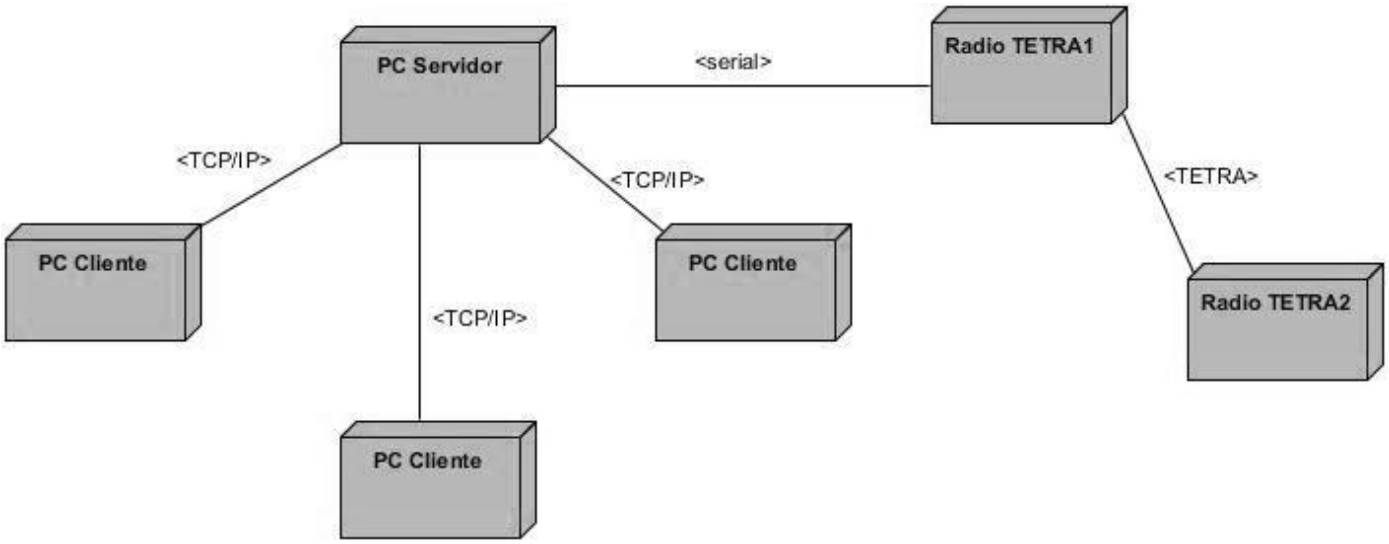

Fig. 9 Diagrama de despliegue.

PC Servidor:

PC donde se encontrará el servidor que permitirá realizar el envío de mensajes de texto y a la cual estarán conectadas las PC clientes.

PC Cliente:

PC donde se encontrará el cliente que recibirá las peticiones para el envío de alarmas al servidor que luego estos se convertirán en mensajes de texto.

Radio TETRA1:

Dispositivo que se conecta a la PC servidor a través del puerto de serie RS232, encargado de enviar el mensaje a su destinatario final.

Radio TETRA2:

Dispositivo que recibe el mensaje a través de la interfaz aire.

4.3 Estilo de código.

La codificación siguiendo un estilo o estándar de código garantiza el desarrollo de un producto siguiendo un estilo de codificación único y uniforme. El estilo de código utilizado, fue definido al comienzo de la implementación del SCADA desarrollado en la universidad, a continuación se describe el mismo.

4.3.1 Definición y usabilidad de los nombres.

- Los nombres de las clases son sustantivos singulares.
- Los nombres deben reflejar el qué y no el cómo.
- Escoger nombres lo suficientemente largo para ser expresivos, pero evitando manejar nombres que dificulten la labor de implantación.
- Evitar nombre que permitan una interpretación subjetiva (evitar ambigüedad y asegurar abstracción).
- Evitar redundancia no repitiendo nombre de clases en sus elementos.
- Dado que los nombres generalmente son el producto de concatenar varias palabras, se debe emplear mayúscula para el inicio de cada palabra y minúscula para el resto de las letras para el caso de los nombres de métodos y funciones. Para el caso de los nombres de variables y atributos debe aplicarse la misma convención, con excepción de la primera letra del nombre, la cual debe ser en minúscula.
- Minimizar el uso de abreviaciones. En caso de ser requeridas, se debe ser consistente en su uso y cada abreviación debe significar solo una cosa. En general agregar a la documentación las abreviaturas.
- Los nombres de los métodos son frases que incluyen verbos.
- Los nombres de los atributos y parámetros son frases con sustantivos.

- Evitar el re-uso de nombres para distinto propósito.

4.3.2 Manejo de errores.

Los errores se pueden manejar mediante mecanismos de excepciones o mediante valores de retorno, aunque esto debe ser uniforme dentro de un mismo objeto. Es buena práctica emplear herramientas para identificar errores en la codificación en caliente.

En esta aplicación el manejo de errores se hace a partir de excepciones. Una excepción es un evento que ocurre durante la ejecución de un programa y detiene el flujo normal de la secuencia de instrucciones del programa. El control de dichas excepciones se utiliza para la detección y corrección de errores. Si hay un error, la aplicación no debería "morirse". Para manejar las se actúa de la siguiente manera:

- Se intenta (try) ejecutar la sentencia o bloque de sentencias que pueden producir algún error.
- Se captura (catch) las posibles excepciones que se hayan podido producir, ejecutando una serie de sentencias que informen o intenten resolver el error.

4.3.3 Documentación y comentarios.

Se documentó el código siguiendo el estilo del SCADA:

- En el código debe documentarse en forma explicativa los pasos que se van ejecutando.
- Emplear oraciones completas al documentar código.
- Documentar mientras se programa.
- Documentar cualquiera cosa que no sea obvia en el código.
- Documentar eliminación de errores y cambios sobre el código.
- Al modificar el código se deben actualizar todos los comentarios y documentación asociada.
- Documentar cada rutina agregando: nombre del desarrollador, fecha, parámetros de entrada, valores de retorno, precondiciones, poscondiciones, dependencia con otros métodos o funciones y descripción general del algoritmo. Además, de realizarse cambios al código, debe indicarse el nombre de la persona que realizó el cambio, la fecha y la descripción del cambio, comenzando desde el o los cambios más recientes.
- Evitar agregar comentarios al final de líneas de código, salvo en el caso de declaraciones. En este caso tales comentarios deben estar alineados.
- Antes de la entrega de la aplicación, eliminar todos los comentarios superfluos y/o temporales con la finalidad de evitar confusiones en su mantenimiento.

4.3.4 Codificación.

- Alinear verticalmente llaves de apertura y cierre.
- Evitar colocar más de una sentencia por línea.
- Minimizar el alcance de las variables para evitar confusión y facilitar el mantenimiento.
- Emplear cada variable y rutina solo para un propósito.
- Evitar el uso de variables públicas, sustituirlas por variables privadas y métodos que provean el valor de tal variable, para mantener el encapsulamiento.
- Emplear select-case o switch en sustitución de if anidados sobre la misma variables.
- Liberar apuntadores de manera explícita.
- 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.
- Inicializar todas las variables.
- Evitar prácticas que incrementan explosivamente la complejidad, como lo son: objetos y variables globales y saltos tipo go-to.

4.4 Pruebas

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. Las pruebas de unidad son la primera fase de las pruebas que se le aplican a cada módulo de un software de manera independiente. Su objetivo es verificar que el módulo, entendido como una unidad funcional de un programa independiente, está correctamente codificado.

Existen distintas técnicas de pruebas que proporcionan criterios para generar casos de pruebas que provoquen fallos en los programas, estas técnicas se agrupan en: técnicas de caja blanca o estructurales y técnicas de caja negra o funcionales, la primera se basa en un minucioso examen de los detalles procedimentales a evaluar, por lo que es necesario conocer la lógica del programa, sin embargo la segunda se basa en la realización de pruebas sobre las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar. (19)

Al sistema propuesto se le realizaron pruebas de caja negra con el objetivo de medir la funcionalidad operativa del software.

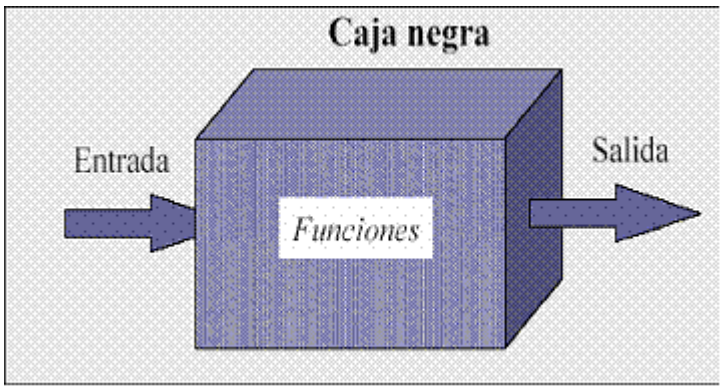

Fig. 10 Representación de las pruebas de caja negra.

4.4.1 Pruebas de caja negra.

Las pruebas de caja negra, también denominadas de comportamiento, se centran en los requisitos funcionales del software. Estas pruebas le permiten al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de dato externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación. (19)

A diferencia de las pruebas de caja blanca, que se basan en la lógica interna del software, las pruebas de caja negra se concentran en su funcionalidad, por lo que mucho del trabajo se realiza interactuando con la interfaz del software. Los casos de prueba generados en este enfoque, se diseñan a partir de valores de entrada y salida. De esta forma, se puede determinar la validez de una salida para un conjunto de entradas proporcionadas.

Existen varias técnicas para la realización de estas pruebas, para realizar las pruebas al módulo para el envío de mensajes, se utilizó la técnica de Partición de equivalencia, donde se identifican clases de equivalencia válida e inválida.

Técnica Partición de equivalencia.

Este método de prueba de caja negra divide el dominio de entrada de un programa en clases de datos, a partir de las cuales deriva los casos de prueba. Cada una de estas clases de equivalencia representa a un conjunto de estados válidos o inválidos para las condiciones de entrada. El primer paso es identificar clases de equivalencia válidas e inválidas. (20)

3.5.1 Ambiente de prueba.

Para la correcta realización de las pruebas, se contó con una serie de recursos que facilitaron el trabajo de ejecutar cada caso de prueba sobre la notificación de alarmas por mensajes de texto.

- ✓ **Recursos Físicos**

Las computadoras utilizadas para el desarrollo de las pruebas contaron con 1.0 giga byte de memoria RAM, microprocesador Intel Core2Duo E4500 con velocidad de 2.20 GHz, motherboard Intel y una capacidad en disco duro de 160 gigas.

Los radios TETRA, HTT-500 y MDT-400.

HTT-500: Nuevo portátil TETRA es un terminal moderno y potente, su diseño ha incorporado los últimos avances tecnológicos, es compatible con más de 18 horas de funcionamiento continuo. Es todo acerca de la cobertura, calidad de audio y la fiabilidad. (21)

Fig. 11 Radio HTT-500

MDT-400: Tecnología digital para las comunicaciones profesionales. Único en su clase, proporciona la cobertura y la versatilidad que hace la diferencia. Un notable diseño flexible que se puede adaptar para aplicaciones complejas tales como la telemetría. (22)

Fig. 12 Radio MDT-400

✓ **Recursos Lógicos**

El sistema operativo en el cual se desarrollaron las pruebas fue Debian 6.0.5 (Squeezee), Kernel 2.6.32-5-686, GNOME 2.30.2.

La velocidad de la red con que se contó fue de 100 megabits por segundo.

4.4.2 Diseño de Pruebas.

En este epígrafe se muestran las pruebas realizadas para demostrar el funcionamiento de la aplicación y los resultados obtenidos en ellas. La intensión de los casos de prueba es probar el sistema de una forma detallada, incluyendo las entradas con las que se experimentarán, las condiciones bajo las cuales se realizan y los resultados esperados.

CPR1. CONEXIÓN CLIENTE-SERVIDOR					
OBJETIVO					
Verificar la conexión correcta entre el cliente y el servidor.					
DESCRIPCIÓN					
El caso de prueba permite comprobar la conexión del cliente y el servidor a partir del ip y el puerto de ambos.					
CONDICIONES PREVIAS					
PROCEDIMIENTO					
Escenario	**Variable1**	**Variable 2**	**Respuesta del**	**Resultado**	**Observa**

	Ip	Puerto	sistema	obtenido	ciones
Conexión Correctamente	*Válido* Cliente: 127.0.0.1 Servidor: 127.0.0.1	*Válido* Cliente: 9998 Servidor:9998	El cliente envía la solicitud.	El cliente envía la solicitud.	
Conexión Incorrectamente	*Válido* Cliente: 127.0.0.1 Servidor: 127.0.0.1	*Inválido* Cliente: 9998 Servidor:4586	El cliente envía la solicitud.	Falla la conexión y no se envía la solicitud.	
	Inválido Cliente: 127.0.0.1 Servidor: 127.0.0.56	*Válido* Cliente: 9998 Servidor:9998			

Tabla 3. Caso de prueba Conexión cliente-servidor.

CPR2. ENVIAR MENSAJE DE TEXTO
OBJETIVO
Verificar que se envíe el mensaje correctamente.
DESCRIPCIÓN
El caso de prueba permite comprobar el envío de mensajes de texto a partir de los servicios que brinda la tecnología ICE, y a través de las redes TETRA.
CONDICIONES PREVIAS
Se ha conectado correctamente el cliente y el servidor. Se encuentra en ejecución la aplicación para el envío de SDS.

PROCEDIMIENTO					
Escenario	**Variable1 Número**	**Variable 2 Mensaje**	**Resultado esperado**	**Resultado obtenido**	**Observaciones**
Enviar Mensaje Correctamente	*Válido* 2020	*Válido* Ocurrencia de alarma1.	Recibir un mensaje con la notificación de la ocurrencia de la alarma.	El operador que se encuentra en el campo recibe el mensaje, y el emisor recibe la confirmación.	
Enviar Mensaje Incorrectamente	*Válido* 2020	*Inválido* Mensaje muy largo.	Recibir un mensaje con la notificación de la ocurrencia de la alarma.	No llega el mensaje, la aplicación cliente recibe una notificación de que el mensaje no se envió.	
	Inválido #@sd	*Válido* Ocurrencia de alarma2.			

Tabla 4. Caso de prueba Enviar mensaje de texto.

En una primera iteración de las pruebas los resultados arrojaron las siguientes no conformidades que fueron resueltas en una segunda iteración:

- Al correr el cliente sin estar corriendo el servidor, no se mostró el mensaje error en la conexión.
- Al poner incorrectamente el número del radio al que se desea enviar el mensaje no se mostró ningún error, solo que el cliente no envía ningún mensaje.
- Al ocurrir algún error en el envío del mensaje no se mostró el mensaje error que se esperaba.

4.4.3 Resultados de las pruebas.

Al módulo desarrollado se le practicaron un conjunto de pruebas para validar las funcionalidades del mismo. Inicialmente se realizaron las pruebas para verificar la correcta conexión entre el cliente y el servidor, para eliminar los posibles errores que podían ocurrir. Después de una correcta conexión, se pasó a probar la principal funcionalidad que es la del envío de mensajes SDS, probando todos los escenarios

en los que podía fallar la misma, y se obtuvo el resultado esperado para todos los casos, enviándose los SDS correctamente. El módulo se probó independientemente del SCADA, mostrando lo genérico que puede ser el mismo, y como no cuenta con una interfaz gráfica pues todo está realizado a nivel de consola puede ser adaptado a cualquier sistema sin ningún inconveniente. De forma general, las pruebas realizadas tuvieron resultados satisfactorios, concluyendo que el módulo se encuentra listo para ser integrado con el SCADA.

4.5 Conclusiones parciales.

En este capítulo se exponen los elementos que indican cómo ha sido implementado el sistema. Se realizaron los diagramas de componentes para modelar la vista estática del sistema, y el diagrama de despliegue para indicar la situación física de los componentes lógicos desarrollados. Mediante este capítulo se definió la organización del código, se implementaron todas las funcionalidades definidas para el sistema y se integraron los resultados en un sistema ejecutable. Con el objetivo de comprobar el correcto funcionamiento de los requerimientos del sistema se han realizado las pruebas que determinaron la validez del software.

CONCLUSIONES

En el presente trabajo de diploma se presentó la investigación y posterior proceso de desarrollo del módulo para la notificación de alarmas por mensajes de texto para el SCADA UX a través de redes TETRA, el cual funciona como intermediario para la notificación de las alarmas generadas por el SCADA a los operadores del sistema. De esta forma se le da cumplimiento al objetivo general de la investigación presentada y se pudo arribar a las siguientes conclusiones:

- TETRA es el estándar más adecuado para la solución pues es el más apropiado en entornos profesionales y sobre todo industriales por las características de los terminales TETRA y además el CEDIN cuenta con las posibilidades tecnológicas.
- Para la construcción y desarrollo del sistema se utilizaron las herramientas y tecnologías seleccionadas como son: lenguaje de programación C++, la herramienta de modelado utilizada fue UML y además se desarrolló el sistema en base a la metodología de desarrollo RUP que permitió desarrollar un software de alta calidad y ayudó a controlar el desarrollo del software a lo largo de todo el proceso.
- Fueron identificados los requerimientos y especificados los casos de uso, modelando la solución y describiendo las funcionalidades del sistema.
- El módulo realizado cuenta con una arquitectura cliente-servidor, a partir de la tecnología ICE, pero implementado de manera flexible para un futuro cambio de tecnología.
- Quedó elaborado el modelo de diseño lo que permitió obtener una abstracción más cercana a la implementación del sistema.
- Se obtuvo una aplicación funcional que responde a los requisitos planteados.
- Se le realizaron pruebas al módulo, las cuales fueron satisfactorias, lográndose de esta manera el cumplimiento de los requisitos exigidos inicialmente, demostrando además la genericidad del módulo para poder ser utilizado en otros entornos, y en otros sistemas.

RECOMENDACIONES

Tomando como base la investigación realizada y los resultados obtenidos durante la realización de este trabajo, se recomienda lo siguiente:

- Utilizar el módulo para la notificación de alarmas por mensajes de texto que se ofrece para la próxima versión del SCADA.
- Implementar el resto de los servicios de datos que ofrece el estándar TETRA.
- Agregarle a la implementación una funcionalidad que permita mostrar las causas por la que pudo fallar el mensaje.

REFERENCIAS BIBLIOGRÁFICAS

1. *Scadas.* [Online] http://www.automatas.org/redes/scadas.htm.

2. **Aragón Cáceres, José A. and Llanes Jiménez, Beatriz.** *Servicio de Integración con Terceros para el Acceso a Variables del sistema SCADA Guardián del ALBA. .* La Habana : s.n., 2009.

3. *Base de Datos de Tiempo Real en el SCADA UX.* La Habana : s.n., 2011.

4. *Proyecto Técnico SCADA UX.* La Habana : s.n., 2012.

5. *Estándar GSM.* [Online] http://es.kioskea.net/contents/telephonie-mobile/gsm.php3.

6. *GPRS-EcuRed.* [Online] http://www.ecured.cu/index.php/GPRS.

7. *Willtek - Tecnologías - UMTS/WCDMA.* [Online] http://www.willtek.com/spanish/technologies/umts_wcdma.

8. **Serrano, Esther Martín.** *Análisis de aplicaciones en entornos de seguridad y emergencias en red TETRA.* 2006.

9. *TETRA + Critical Communications Association.* [Online] http://www.tetramou.com/.

10. *CAPÍTULO 2 ANÁLISIS DEL ESTÁNDAR TETRA.* [Online] http://dspace.epn.edu.ec/bitstream/15000/8669/5/T%2011171%20CAPITULO%202.pdf.

11. *Redes de Acceso Celular: Introducción.* [Online] www.ea1uro.com/pdf/RedesPMRSistemaTETRA.pdf.

12. *Debian caracteristicas.* [Online] http://www.slideshare.net/wolf_dragons/debian-caracteristicas.

13. Lenguaje UML | TECHFICO - Cuaderno de Infomatica. [Online] abril 2010. http://techfico.blogspot.com/2010/04/lenguaje-uml_23.html.

14. Free Download Manager. [Online] www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(MÍ)_14720_p/.

15. Breve introduccion a c++. [Online] noviembre 2009. http://wifi4.blogspot.com/2009/11/breve-introduccion-c_29.html.

16. Eclipse SDK - Epistemowikia. [Online] http://campusvirtual.unex.es/cala/epistemowikia/index.php?title=Eclipse_SDK.

17. *Arquitectura Cliente Servidor - EcuRed.* [Online] http://www.ecured.cu/index.php/Arquitectura_Cliente_Servidor.

18. **Larman, Craig.** *UML y Patrones. .*

19. *Flujo de Trabajo de Prueba.* Universidad de las Ciencias Informaticas. Cuba : s.n., 2007.

20. *técnicas de prueba.* [Online] http://indalog.ual.es/mtorres/LP/Prueba.pdf.

21. *Teltronic S.A.* [Online] http://www.teltronic.es/sites/default/files/HTT-500_en.pdf.

22. *Teltronic S.A.* [Online] http://www.teltronic.es/sites/default/files/MDT-400%20English.pdf.

23. *Object Management Group - UML.* [Online] 2012. http://www.uml.org/.

24. *Introducción al Módulo HMI. Modo Ejecución.* La Habana : s.n., 2008.

25. **Pressman, Roger S.** *Ingeniería de Software, un enfoque práctico.* Quinta edición. ISBN: 8448132149..

26. **Larman, Craig.** *UML y Patrones. .*

27. *Flujo de Implementación.* Universidad de las Ciencias Informáticas. Cuba. : s.n., 2007.

28. **Pressman, Roger.** *"Ingeniería de software. Un enfoque práctico".*

29. *Flujo de Trabajo de Prueba.* 2007. Universidad de las Ciencias Informaticas. Cuba.

30. **Stavroulakis, Peter.** *"TErrestrialTrunked RAdio-TETRA. A global Security Tool".* 2007.

31. **BOOCH, GRADY, JACOBSON, IVAR, RUMBAUGH, JAMES.** *El Lenguaje Unificado de Modelado. Manual de Referencia.* 2007.

32. **PRESSMAN, Roger S.** *"Ingeniería de Software. Un enfoque Práctico". 6ta edición.* 2007.

33. **SOMMERVILLE, I.** *"Ingeniería del Software". 7ma Edición.* 2005.

ANEXOS

Vista Significativa del código.

Método sendSDS

```
int TETRASDSSender::sendSDS(string sds, string ssiDestination)
{
    unsigned char commandConfigSDS[] = "\rat+ctsd=12,0,0\r";
    unsigned char commandPEIControler[] = "\rat+pcmf=r\r";
    unsigned char commandEcho[] = "\rate0\r";

    serialIO->serialWrite(commandEcho, strlen((char*)commandEcho));
    usleep(1000);
    serialIO->serialWrite(commandPEIControler, strlen((char*)commandPEIControler));
    usleep(1000);
    serialIO->serialWrite(commandConfigSDS, strlen((char*)commandConfigSDS));
    usleep(1000);

    unsigned messageSize = 2 + sds.length();

    char callConfig[50];
    memset(callConfig, 0, 50);
    strncpy(callConfig, "\rat+cmgs=", 9);
    unsigned ssiLength = ssiDestination.length();
    strncpy(&callConfig[9], (const char*)ssiDestination.c_str(), ssiLength);

    char charMessageSize[6];
    memset(charMessageSize, 0, 6);
    unsigned numBits = messageSize * 8; // From bytes to bits
    iToA(numBits, charMessageSize);
    strncpy(&callConfig[9 + ssiLength], ",", 1);
    strncpy(&callConfig[10 + ssiLength], charMessageSize, strlen(charMessageSize));
    strncpy(&callConfig[10 + ssiLength + strlen(charMessageSize)], "\r", 1);

    serialIO->serialWrite((unsigned char*)callConfig, strlen((char*)callConfig));

    unsigned char message[256];
    message[0] = 0x2;
    message[1] = 0x1;

    for (unsigned i = 0; i < messageSize - 2; ++i)
    {
        message[i+2] = charToHex(sds[i]);
    }
    message[messageSize] = '\r';
```

```
    serialIO->serialWrite(message, messageSize + 1);

    return 1;
}
```

Conexión con el Puerto de serie.

Clase Serial.

```
static void * readSerialPortThread( void *)
{
        char buf;
        while (1)
        {
                if(read(fdStatic, &buf, 1) > 0)
                {
                        cout << buf << endl;
                }
                usleep(10000);
        }
}

int Serial::openDevice(char * device)
{
        struct termios oldtio,newtio;
        int fd;

        char dev[20];

        snprintf( dev, sizeof(dev), device, 0 );

        /* open the device to be non-blocking (read will return immediately) */
        fd = open( dev, O_RDWR | O_NOCTTY | O_NONBLOCK );
        if ( fd < 0 )
        {
                perror( dev );
                exit( -1 );
        }

        tcgetattr( fd,&oldtio ); /* save current port settings */
        /* set new port settings */
        /* see 'man termios' for further settings */
        newtio.c_cflag = BAUDRATE | CLOCAL | CREAD | CS8;

        newtio.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR | IGNCR | ICRNL);
```

```
        newtio.c_oflag &= ~OPOST;
        newtio.c_lflag &= ~(ECHO | ECHONL | ICANON | ISIG | IEXTEN);
        newtio.c_iflag &= ~(IXON | IXOFF | IXANY);

        newtio.c_cflag |= CS8;

        newtio.c_iflag = 0;
        newtio.c_oflag = 0;
        newtio.c_lflag = 0;
        newtio.c_cc[VMIN] = 1;  /*Read 1 byte at a time, no timer*/
        newtio.c_cc[VTIME] = 0;

        tcsetattr( fd, TCSANOW, &newtio );
        tcflush( fd, TCIOFLUSH );

        fdStatic = fd;
        return fd;
}

int Serial::serialWrite(unsigned char *buf, unsigned size)
{
        unsigned aux = 0;
        int w = 0;
        while (aux < size)
        {
                w = write(fdStatic, buf + aux, size - aux);
                if(w != -1)
                {
                        aux += w;
                }
        }
    return 0;
}

int Serial::serialRead()
{
        pthread_create(&thread, NULL, readSerialPortThread, NULL);
        return 0;
}
```