UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 5

“Herramienta de simulación de datos en consolas HMI para el SCADA Guardián del Alba”.

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas.

Autor: Emilio Bejar Suárez

Tutor: Ing. Raudi Agdel Bacallao Sánchez

La Habana, 2015

“El éxito consiste en vencer el temor al fracaso”.

Charles Augustin Sainte.

DECLARACIÓN DE AUTORÍA

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

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

_____

Firma del autor

Emilio Bejar Suarez

_____

Firma del tutor

Ing. Raudi Agdel Bacallao

DATOS DE CONTACTO

Tutor: **Ing. Raudi Agdel Bacallao Sánchez**

Edad: 29 años

Institución: Universidad de las Ciencias Informáticas (UCI)

Cargo: Especialista

Título: Ingeniería en Ciencias Informáticas

E-mail: rbacallao@uci.cu

Dedicatoria

Quiero dedicar pocas palabras pero sinceras a las personas más importantes en mi vida y por ende, a las que les agradezco mi existencia hoy como persona , en primer lugar a mi abuela que en donde quiera que esté refunfuñando sepa que ya cumplí su sueño. A mi mamá por darme el ánimo siempre en el momento exacto y en el no exacto, de ambas formas, aportó para llegar aquí. A mi papá por ser la persona que siempre quiero tratar de hacer sentir orgulloso. A mi hermana pues estoy orgulloso de tenerla como hermana, a mi novia por quererme, por preocuparse y ayudarme en una año que no es difícil pero si obstinante. A mi cuñado que siempre está listo para dar hit en el momento exacto y a toda mi familia que siempre me ha querido y ayudado. También dedicárselas a todos mis compañeros y amigos del aula 5501 como Alberto, Alfonso, Carlos y Dayana y del apartamento antiguo 87104 a Dariel Costa, Yasser, y Javier Cuevas. Y por último, a mis mejores amigos que siempre estarán presentes Aldo, Sergio y Michelito.

Agradecimientos

Agradecer en primer lugar a mis tutores Bernardo Zaragoza Hijuelos y Raudi Agdel Bacallao Sánchez por de alguna manera ayudarme a empezar y terminar este trabajo; a los choferes de la Uci, en especial al de la guagua 747 por llevarme y traerme casi todos los días. Al Ing. José Antonio Aragón que su opinión para mí siempre valía oro, a mi oponente el Ing. Yosvani Ramírez, a la profe Msc. Mariela, al jefe de tribunal Msc. Fermín, al tutor que tuve durante 1 semana el Ing. Ridel y en especial a todos los profesores que tuve durante 5 años de carrera, pues cada uno aportó un grano de arena para lograr mi título de hoy y por último, a todos mis amistades de la UCI, las que nunca olvidaré.

Resumen

El módulo Interfaz Hombre-Máquina es el que permite el contacto directo con el sistema a través de sinópticos o mímicos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado en el SCADA GALBA, integrando los datos que se emiten por los demás componentes. De esta manera, el HMI posibilita un mayor acercamiento al proceso supervisado, pero en ocasiones, en dicho proceso, no se cuenta con los datos directos de los dispositivos de campos para comprobar si los requerimientos o funcionalidades fueron bien implementados. En el ambiente de prueba del HMI se hace necesario el uso de valores reales o al menos, lo más exactos posible. Para dar solución a dicho problema se realiza el siguiente trabajo que tiene como objetivo desarrollar una solución que provea a la Interfaz Hombre-Máquina del GALBA de una herramienta que simule el comportamiento de variables analógicas y digitales permitiendo el análisis de los procesos de supervisión.Para la solución de dicho problema se seleccionó como plataforma de desarrollo el sistema operativo Debian 7, como framework de desarrollo QT, como lenguaje de programación C++ y XML, como entorno de desarrollo Qt Creator, como biblioteca gráfica Qwt y como lenguaje de modelado UML, como herramienta case Visual Paradigm y como metodología de desarrollo AUP en su variante UCI. Como resultado del trabajo se desarrolló una herramienta para simular la adquisición de datos en el módulo de visualización del SCADA GALBA.

Palabras clave: SCADA, módulos, variables, procesos, dispositivos, simulador

Índice de Contenido

Índice de Tablas

Introducción

En Cuba se han desarrollado numerosas aplicaciones informáticas para contribuir a desarrollar la economía, así como la informatización de empresas tanto del país como del exterior. Una de las instituciones que ha dado grandes pasos en este sentido es la Universidad de las Ciencias Informáticas (UCI). En esta universidad se encuentra el Centro de Informática Industrial (CEDIN) que, conjuntamente con la empresa Petróleos de Venezuela SA (PDVSA), desarrolla el sistema integral de supervisión de procesos SCADA Guardián del ALBA (GALBA).

Los sistemas de supervisión, control y adquisición de datos, constituyen el núcleo de la mayoría de los sistemas automatizados, por tal motivo se erigen como altamente estratégicos para cualquier industria. Un SCADA es un software especialmente diseñado para funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo y controlando el proceso que se está realizando desde la pantalla del ordenador. Además, provee a diversos usuarios información generada en el proceso productivo, control de calidad, supervisión y mantenimiento.

Por la magnitud de los sistemas SCADA, estos son desarrollados empleando generalmente arquitecturas orientadas a componentes y distribuidas, que permiten entre otras cosas, distribuir la carga de trabajo del sistema en diferentes ordenadores y dividir el problema de desarrollar un sistema SCADA en varias partes más pequeñas. En el caso del sistema SCADA GALBA, el mismo esta constituido por diferentes módulos, entre los que se encuentran: configuración, seguridad, adquisición de datos, comunicaciones e interfaz hombre-máquina (HMI).

El módulo de interfaz hombre-máquina del SCADA GALBA es el encargado de la interacción del usuario con el mismo a través interfaces gráficas. Este se encuentra dividido en un ambiente de edición de la configuración y un ambiente de ejecución donde se pueden supervisar y controlar los procesos. En el proceso de desarrollo y pruebas del módulo HMI es necesario contar con datos lo más cercanos posibles al proceso que va a monitorizar el SCADA para obtener como resultado un sistema con altos estándares de calidad y de satisfacción del usuario.

En el proceso de desarrollo y prueba del SCADA GALBA en el módulo de visualización se usan simuladores de datos como el ModSim,el ModbusPal y el DriverDemo teniendo como limitantes:

1. En tiempo de pruebas es difícil capturar los datos físicos de dispositivos puesto que las áreas de desarrollo y supervisión están distantes entre sí geográficamente.
2. Los simuladores usados se conectan a nivel de driver y tienen una conexión directa en ejecución con todos los módulos del SCADA GALBA.
3. Las simulaciones utilizadas no generan el conjunto de datos con las potencialidades que se necesitan como permisos de lectura y escritura, además no son lo suficientemente flexibles como para procesar de forma automática los datos en estampas de tiempos, la accesibilidad, el rango numérico y calidad de los mismos.

De todo lo expuesto anteriormente se presenta como **problema a resolver:**

¿Cómo garantizar la disponibilidad de datos en el proceso de prueba del ambiente de visualización del HMI?

A partir del problema de investigación planteado se define como **objeto de estudio** las herramientas de simulación numérica y como **campo de acción** las herramientas de simulación numérica en los HMI.

El objetivo general consiste en: Desarrollar una herramienta para simular la adquisición de datos en el módulo de visualización del SCADA GALBA.

Para dar cumplimiento al objetivo planteado, se elaboraron una serie de tareas investigativas que se muestran a continuación:

1. Análisis del estado de los simuladores de datos, así como las herramientas, estándares y métodos para implementar este tipo de aplicación para sistemas SCADA.
2. Análisis de la estructura de los módulos del HMI del SCADA GALBA para comprender su funcionamiento actual.
3. Descripción del prototipo funcional de la herramienta de simulación de datos.
4. Desarrollo del servidor de simulación de datos para el HMI del SCADA GALBA.
5. Desarrollo del cliente de simulación de datos para el HMI del SCADA GALBA.

6. Validación de los resultados a partir de las pruebas realizadas.

Métodos Empíricos

1. **Consulta de fuentes de información:** Este método se utilizó para la consulta de las fuentes bibliográficas durante la investigación, lo cual fue muy importante para la elaboración del marco teórico. A través del mismo se pudo abarcar y proveer gran parte de toda la información referente al tema de los simuladores y su utilización en los sistemas SCADA.
2. **Observación:** Este método se utilizó para apreciar el comportamiento de la herramienta elaborada de acuerdo a los algoritmos implementados en la misma, dando una mejor percepción de los resultados alcanzados y permitiendo de esta manera erradicar los errores cometidos durante la implementación de la aplicación.
3. **Pruebas:** Método utilizado para validar los resultados obtenidos con la solución propuesta a partir de la investigación realizada. A través del mismo se pudo obtener una mejor percepción de lo que se estaba elaborando, haciendo una comparación entre los resultados esperados y la respuesta del sistema al realizar las pruebas.

Métodos teóricos

Analítico-Sintético: Este es un método teórico que se utilizó para extraer y analizar la información acerca de los sistemas de simulación, ya que se hacía necesaria una amplia investigación sobre dichos sistemas para lograr un mejor entendimiento de sus funcionalidades, un mayor acercamiento al estudio de estas herramientas y su gran utilidad para obtener resultados aproximados.

Este documento está estructurado en tres capítulos que recogen todo el proceso de desarrollo de este trabajo

CAPÍTULO 1: Fundamentación teórica y evaluación de las tecnologías.

En este capítulo se refleja las principales características y conceptos asociados a los simuladores y sistemas SCADAS, además se describen las principales tecnologías que se emplearán en la solución propuesta.

CAPÍTULO 2: Proceso de análisis y diseño.

En este capítulo se reflejan las actividades realizadas en los procesos de análisis y diseño de la

herramienta de simulación así como la de sus principales artefactos como son las clases de diseño y los requisitos funcionales y no funcionales de la solución propuesta.

CAPÍTULO 3: Implementación y validación de la solución.

En este capítulo se refleja la implementación del sistema en términos de componentes, con el objetivo de mostrar los componentes que se obtienen de implementar las clases y los paquetes definidos en el diseño, además de las pruebas realizadas a la herramienta.

Capítulo 1

Fundamentación teórica y evaluación de las tecnologías.

1.1 Introducción

En el presente capítulo se introducen las principales características y conceptos asociados a los simuladores. Para ello se abordan definiciones, importancia y ventajas de estos sistemas. Se realiza una breve descripción de los sistemas SCADA, de sus distintos módulos y ejemplos de simuladores utilizados por estos. Se describen las principales tecnologías que se emplean en la construcción de software para la simulación.

1.2 Sistemas SCADA

Los sistemas SCADA utilizan la computadora y tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales. Estos sistemas son partes integrales de la mayoría de los ambientes industriales complejos o distantes geográficamente (ver Figura 1), ya que pueden recoger la información de una gran cantidad de fuentes rápidamente y la presentan a un operador de forma amigable. Los sistemas SCADA mejoran el proceso de monitoreo y control proporcionando la información oportuna para poder tomar decisiones operacionales apropiadas. (Montero,2010)

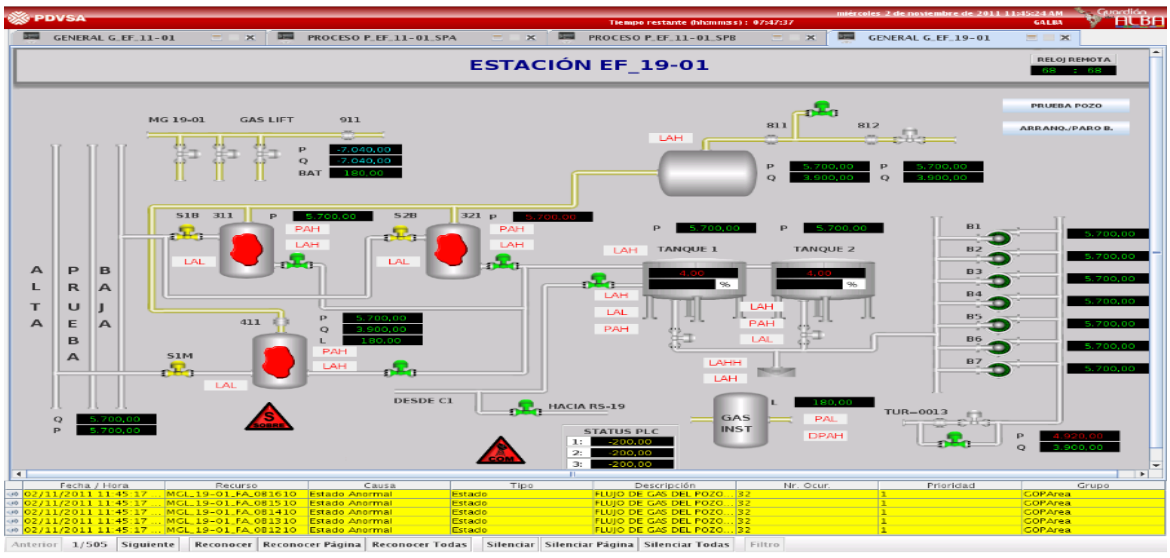

Figura 1: Sistema SCADA

Los primeros sistemas automatizados SCADA fueron altamente modificados con programas de aplicación específicos para atender a requisitos de algún proyecto particular. Varios ingenieros asistieron al diseño de estos sistemas y realizaron un análisis de las características de sus industrias para luego integrar estos sistemas al proceso productivo de acuerdo a los intereses de cada institución.

Los proveedores de sistemas SCADA, deseando reutilizar su trabajo previo sobre nuevos proyectos, perpetuaron esta imagen de industria específica por su propia visión de los ambientes de control con los cuales tenían experiencia. Solamente cuando nuevos proyectos necesitaron funciones y aplicaciones adicionales, entonces los desarrolladores tuvieron la oportunidad de adquirir experiencias en otras industrias.

En la actualidad, estos proveedores están diseñando sistemas que son pensados para resolver las necesidades de muchas industrias, con módulos de software industriales específicos disponibles para proporcionar las capacidades requeridas comúnmente. Usualmente se encuentran software de este tipo comercialmente disponible adaptado para procesamiento de papel y celulosa, industrias de aceite y gas, hidroeléctricas, gerencia y provisión de agua, así como el control de fluidos. (Montero, 2010)

En la Figura 2 se muestra el esquema básico de un SCADA conectado a un proceso automatizado, y para su mejor comprensión una breve explicación de sus componentes. (Montero, 2010)

Figura 2: Esquema básico de un sistema SCADA.

1.3 Componentes del sistema SCADA GALBA

Los módulos o bloques de software que permiten las actividades de adquisición, supervisión y control son los siguientes: (Autómatas Industriales, 2000)

Configuración: Este componente permite al operador configurar los procesos, definir y gestionar las variables, editar los controladores, los comandos, las alarmas y variadas opciones adicionales. Dicho componente permite a los mantenedores definir el ambiente de trabajo adaptándolo mejor a la aplicación en particular que se desea desarrollar.

Interfaz Hombre-Máquina: Este proporciona al operador las funciones de control y supervisión sobre la planta, permitiéndole de esta forma un contacto directo con el sistema. Representa los procesos que ocurren en el campo en tiempo real, los componentes implicados, los sensores, las estaciones remotas y el sistema de comunicación, dándole al operador total control. Estos procesos son representados mediante sinópticos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado e importados desde otra aplicación durante la configuración del paquete

Adquisición: Presenta como principales características la adquisición de datos del nivel de recolección en tiempo real, la conversión de unidades, el procesamiento de variables calculadas, la detección y el manejo de alarmas, el control de calidad de los datos recolectados, la publicación a los clientes de la actualización de los puntos y la sucesión de las alarmas y los eventos. Ejecuta comandos sobre toda la información almacenada a nivel de campo.

Base de datos histórica: Este se encarga del almacenamiento y procesamiento ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.

Comunicación o Middleware: Este módulo tiene como finalidad proporcionar la capa de comunicación de alto nivel, tanto sincrónica, como asincrónica, para la comunicación de todos los módulos que conforman el sistema.

1.4 Introducción a los simuladores

Las primeras referencias sobre simulación se encuentran hacia el año 1940, cuando Von Neumann y Ullman trabajaron en la simulación del flujo de neutrones para la construcción de la bomba atómica en el proyecto Montecarlo.

Desde entonces se conocían las técnicas de simulación como procesos Montecarlo, aunque en la actualidad se diferencian ambas cosas, siendo los segundos un tipo particular de simulación. También se realizó un proceso de simulación para el proyecto APOLO dentro del plan espacial de la N.A.S.A, acerca del movimiento dentro de la atmósfera de la luna.

Los simuladores son objetos de aprendizaje que mediante un programa de software, intentan modelar parte de una réplica de los fenómenos de la realidad y su propósito es que el usuario construya conocimiento a partir del trabajo exploratorio y la inferencia por descubrimiento. Estos se desarrollan en un entorno interactivo, que permite al usuario modificar parámetros y ver cómo reacciona el sistema ante el cambio producido. Tienen sus orígenes en la teoría de muestreo estadístico y análisis de sistemas físicos probabilísticos complejos. El aspecto común de ambos es el uso de números y muestras aleatorias para aproximar soluciones.(Automatas Industriales, 2000).A continuación, en la figura 3 se muestra un ejemplo de simulador de variables analógicas y digitales para un sistema SCADA.

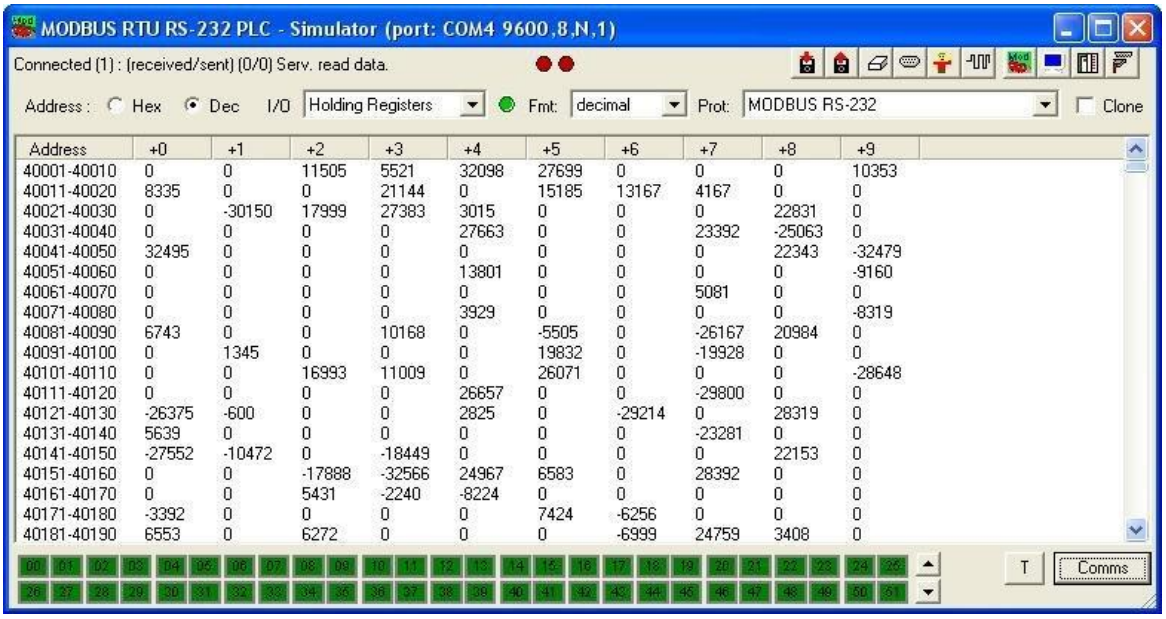

Figura 3: Simulador de variables para un SCADA.

Es importante señalar dos conceptos fundamentales referentes a dos de los tipos de variables más usados en un sistema SCADA.

1.4.1 Variable analógica

Una variable analógica es aquella que toma infinitos valores entre dos puntos cualesquiera de la misma. Un ejemplo de variable analógica es la temperatura. Si se analiza su variación durante un día, se observa que no puede pasar de un valor a otro dando un salto. Esto quiere decir que si la temperatura se incrementa de 11 grados a 17 grados, toma infinitos valores intermedios. (UAG, 2008)

1.4.2 Variable digital

Las variables digitales son aquellas que pueden tomar un número infinito de valores comprendidos entre dos límites. La mayoría de los fenómenos de la vida real dan señales de este tipo, un ejemplo de variable digital puede ser una alarma lanzada en un sistema informático para indicar alguna anomalía. (UAG, 2008)

Dentro de los tipos de variables digitales se encuentran las variables binarias las cuales toman solamente un estado de dos posibles. Algunos ejemplos son: una lámpara encendida o apagada, una llave cerrada o abierta, el abrir y cerrar una válvula, una compuerta, un circuito con tensión o sin tensión, es decir elementos que comúnmente son asociados a pulsos en el ámbito de la electricidad. La importancia de las variables binarias radica en el hecho de que los circuitos internos de las computadoras trabajan con ellas. (UAG, 2008)

1.4.3 Clasificación de los simuladores

Existen diversos criterios según los cuales los simuladores pueden ser clasificados (UAG, 2008). Sobre la base de sus características de diseño y construcción, el simulador puede ser:

1. **Genérico:** Estos no representan un determinado equipo (marca y modelo)
2. **Semi-real:** En este caso el equipo representa un determinado modelo de la vida real, pero se utiliza una réplica que ha sido diseñada ajustándose al mismo que se utiliza en la realidad
3. **Real:** Se utiliza el mismo equipo que se usa en la vida real, simulando dicha operación

La decisión del tipo de simulador a utilizar en un determinado programa, depende de las características del entrenamiento requerido y en todos los casos resulta de máxima importancia, la decisión estratégica de un profesor y/o instructor experto en la materia. Es la forma de incorporar el equipo adecuado en el momento adecuado.

1.4.4 Metodología del proceso de simulación

Planificar un proceso de simulación (UCLA, 2010) requiere de los siguientes pasos:

1. Formulación del problema

En este se definen las cuestiones para las que se buscan las respuestas, las variables implicadas y las medidas de ejecución que se van a usar. Esta fase es muy importante para poder alcanzar un modelo válido.

2. Colección de datos y análisis

Durante este paso se recoge el mayor volumen de datos, se reducen y se analizan. Los métodos de recogida de datos son tan variados como los problemas a los que éstos se pueden aplicar. Si se clasifican por su sencillez, se puede ir desde las aproximaciones manuales hasta las técnicas más sofisticadas de alta tecnología.

3. Desarrollo del modelo

Incluye la construcción y depuración del modelo del sistema real, incluyendo la selección de un lenguaje de programación, codificación del modelo.

4. Verificación y validación del modelo

La verificación del modelo seleccionado consiste en ver cuál es la consistencia interna del mismo mientras que la validación asegura la existencia de la correspondencia entre el sistema real y el modelo. Un buen método para la validación es hacer un test para ver cómo el modelo predice el comportamiento del sistema ante determinadas entradas.

La verificación y validación del modelo se realiza en todos los niveles de modelización: modelo conceptual, modelo lógico y un modelo de ordenador. La verificación se centra en la consistencia interna del modelo, mientras que la validación se interesa por la correspondencia entre el modelo y la realidad.

5. Experimentación y análisis de las salidas

Se han de diseñar los experimentos que se van a llevar a cabo sobre el modelo y luego analizar las salidas obtenidas, de forma que se pueda responder a las cuestiones que se

plantearon.

6. Implantación de los resultados de la simulación

Este paso final es uno de los más importantes y el que más se descuida de todo el proceso. Parece obvio que los beneficios de un largo y costoso análisis no se realizarán sin una implementación apropiada y una aceptación por parte de los usuarios.

1.4.5 Modelos de simulación y sus clasificaciones

Un modelo de simulación es una representación simplificada de la realidad diseñada para representar, conocer o predecir propiedades del objeto real .Estos se construyen con una finalidad: estudiar el objeto real con más facilidad y deducir propiedades difíciles de observar en la realidad. Los modelos de simulación pueden representar objetos o procesos (simulación). (Escolme Virtual, 2010)

Existen múltiples tipos de modelos (Escolme Virtual, 2010) para representar la realidad. Algunos de ellos son:

1. **Dinámicos:** Estos se utilizan para representar sistemas cuyo estado varía con el tiempo.
2. **Estáticos:** Se utilizan para representar sistemas cuyo estado es invariable a través del tiempo.
3. **Matemáticos:** Representan la realidad en forma abstracta de diversas maneras.
4. **Físicos:** Son aquellos modelos en que la realidad es representada por algo tangible, construido en escala o que por lo menos se comporta en forma análoga a esa realidad (maquetas, prototipos o modelos analógicos).
5. **Analíticos:** La realidad se representa por fórmulas matemáticas. Estudiar el sistema consiste en operar con esas fórmulas matemáticas (resolución de ecuaciones).
6. **Numéricos:** Se tiene el comportamiento numérico de las variables intervinientes. No se obtiene ninguna solución analítica.
7. **Continuos:** Representan sistemas cuyos cambios de estado son graduales. Las variables intervinientes son continuas.
8. **Discretos:** Representan sistemas cuyos cambios de estado son de saltos. Las variables varían en forma discontinua.

9. **Determinísticos:** Son modelos cuya solución para determinadas condiciones es única y siempre la misma.

10. Estocásticos: Representan sistemas donde los hechos suceden al azar, lo cual no es repetitivo. No se puede asegurar cuáles acciones ocurren en un determinado instante. Se conoce la probabilidad de ocurrencia y su distribución probabilística.

1.4.6 Ventajas del uso de la simulación

Aunque la técnica de simulación generalmente se ve como un método de último recurso, recientes avances en la metodología de simulación y la gran disponibilidad de software que actualmente existe en el mercado, han hecho que la técnica de simulación sea una de las herramientas más ampliamente usadas en el análisis de sistemas. (Pothamsetty, 2011)

Thomas H. Naylor ha sugerido que un estudio de simulación es muy recomendable porque presenta las siguientes ventajas (Pothamsetty, 2011):

- A través de un estudio de simulación, se puede estudiar el efecto de cambios internos y externos del sistema pues al hacer alteraciones en el modelo del sistema, se pueden ir observando los efectos de esas alteraciones en el comportamiento del propio sistema.
- Una observación detallada del sistema que se está simulando puede conducir a un mejor entendimiento del sistema y por consiguiente, a sugerir estrategias que mejoren la operación y eficiencia del sistema.
- La técnica de simulación puede ser utilizada para experimentar con nuevas situaciones, sobre las cuales tiene poca o ninguna información. A través de esta experimentación se puede anticipar mejor a posibles resultados no previstos.
- Cuando nuevos elementos son introducidos en un sistema, la simulación puede ser usada para anticipar cuellos de botella o algún otro problema que puede surgir en el comportamiento del sistema.
- El tiempo puede ser compensado en los modelos de simulación. El equivalente de días, semanas y meses de un sistema real en operación frecuente pueden ser simulados en solo segundos, minutos u horas en una computadora. Esto significa que un largo número de alternativas de solución pueden ser simuladas y los resultados pueden estar disponibles de forma breve y ser suficientes para influir en la elección de un diseño para

un sistema.

- En simulación cada variable puede sostenerse constante excepto algunas cuya influencia está siendo estudiada. Como resultado el posible efecto de descontrol de las variables en el comportamiento del sistema necesitan no ser tomados en cuenta como frecuentemente debe ser hecho cuando el experimento está desarrollado sobre un sistema real.
- Es posible reproducir eventos aleatorios idénticos mediante una secuencia de números aleatorios. Esto hace posible usar las técnicas de reproducción de varianza para mejorar la precisión con la cual las características del sistema pueden ser estimadas para dar un valor que refleje el esfuerzo de la simulación.

1.4.7 Simuladores utilizados en los sistemas SCADAS

En el mundo existen diferentes simuladores para la generación de valores numéricos con el fin de utilizarlos en pruebas para diferentes procesos en los sistemas SCADA , entre ellos se hace referencia a tres de los más utilizados en el proyecto SCADA GALBA.

MODBUSPAL

ModbusPal es un simulador que reproduce un ambiente realista ,que puede ser modificado para requisitos particulares y controlado por scripts. El núcleo de ModbusPal está escrito en Java, es compatible de forma nativa con TCP/IP y la comunicación serial es compatible si la biblioteca RxTx está instalado en el ordenador. ModbusPal es gratuita y de código abierto, liberado bajo la licencia GPL.

MODSIM (Modbus slave data simulator)

Modsim es un simulador principalmente de Windows que esta diseñado como una herramienta de prueba para la verificación del funcionamiento de un protocolo en sistemas nuevos o existentes. Permite definir varios bloques de elementos de datos que se puede acceder desde varios maestros Modbus conectados a él.(Universidad de Chile, 2010) **Ver figura 4**

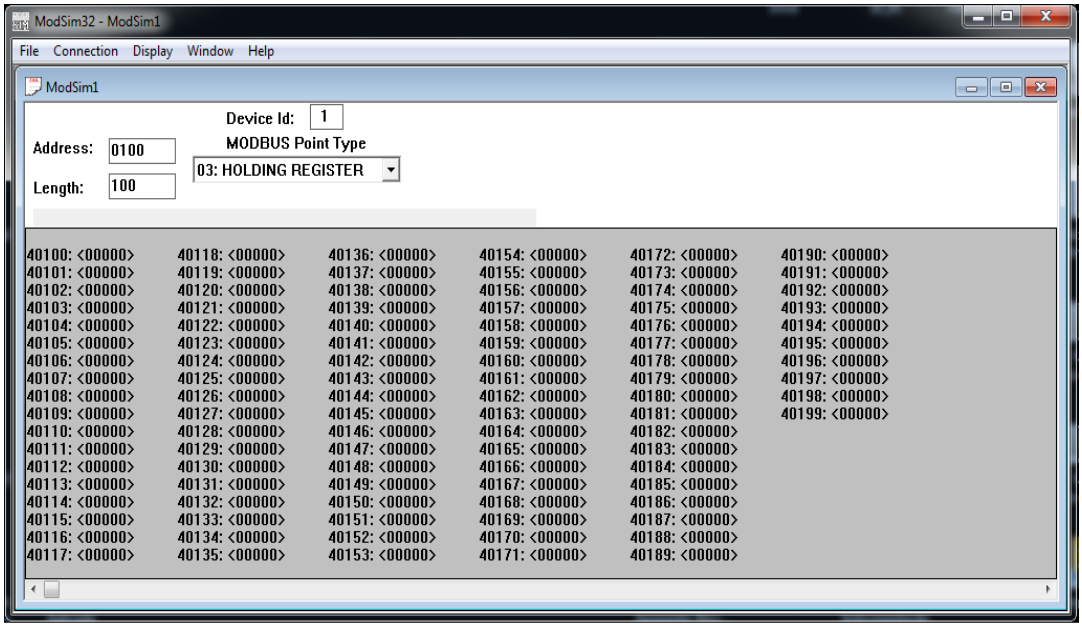

Figura 4: Datos generados por el Modsim.

MPSS / ESN

Es un simulador de sistemas de alimentación modular con sistema de control SCADA y sistema SCADA-NET. Está especialmente diseñado para ser manejado por usuarios a nivel técnico y profesional o el nivel inicial en la educación superior.

El simulador incluye las partes principales de un sistema de energía como: generación, transformación, transporte, distribución y consumo. Incluye varios elementos que juegan un rol muy importante en el control de sistemas de energía y la protección de los mismos, entre ellos se encuentran: el regulador automático de voltaje, el control automático de frecuencia, sincronización manual y automática.(Universidad de Chile, 2010).

1.5 Plataforma de desarrollo (Debian 7 (Wheezy))

El software libre brinda libertad a los usuarios sobre el producto adquirido y por tanto, el mismo puede ser usado, copiado, estudiado, modificado y redistribuido libremente, otorgando de esta forma libertades a los usuarios sobre el software: la libertad de usar el programa con cualquier propósito; de estudiar el funcionamiento del programa y adaptarlo a las necesidades; de distribuir copias, con lo que puede ayudar a otros; de mejorar el programa y hacer públicas las mejoras, de modo que toda la comunidad se beneficie, así mismo ofreciendo el acceso pleno al código fuente sin ningún requisito auxiliar. (GNU Operating System, 2005).

El proyecto Debian es un grupo mundial de voluntarios que se esfuerzan por producir una

distribución de sistema operativo que esté compuesta enteramente de software libre. El producto principal del proyecto a la fecha es la distribución de software Debian GNU/Linux, la cual incluye a Linux como núcleo del sistema operativo así como miles de aplicaciones pre-empaquetadas.

Debian se caracteriza por la disponibilidad en varias plataformas hardware, tiene una amplia gama de software disponible, cuenta con herramientas para facilitar el proceso de instalación y actualización del software, no tiene marcado ningún entorno gráfico en especial, por ende, puede usar GNOME, KDE u otros. Debian es una distribución de gran estabilidad y utilidad por lo que muchos desarrolladores la han tomado para crear otras nuevas distribuciones como: Knoppix, Ubuntu así como también, brinda gran seguridad y robustez además de tener una gran facilidad de uso. (Debian-es, 2010).

1.6 Framework (marco de trabajo) de desarrollo

Los framework de desarrollo son estructuras conceptuales y tecnológicas con soporte definido, normalmente con artefactos o módulos de software concretos; en base a la cual, otro proyecto de software puede ser más fácilmente desarrollado y organizado. Los framework de desarrollo son diseñados con la intención de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requisitos de software que tratando con los detalles de bajo nivel, para proveer un sistema funcional. (QT Digia, 2012)

Qt es un framework multiplataforma, que se utiliza para el desarrollo de aplicaciones, está escrito en C++; sin embargo, es posible utilizarlo con otros lenguajes como C#, PHP, Python, y Ruby. Qt ofrece bibliotecas de código para: creación de interfaces gráficas de usuario, acceso a bases de datos, manipulación de contenido XML (*eXtensible Markup Language*), comunicación en red, visualización con OpenGL, entre otras. Además extiende el lenguaje de programación C++, a través de macros y meta-información (QT Digia, 2012)

1.7 Entorno de desarrollo

Los entornos de desarrollo integrados (IDE) actualmente están considerados como un conjunto de herramientas indispensables para un desarrollador de software y a la vez, pueden ser utilizados para uno o varios lenguajes de programación. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. (GNU Operating System, 2015)

Pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes. Proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Visual Basic, Object Pascal, etc. Otra característica importante es que pueden funcionar como un sistema en tiempo de ejecución en algunos lenguajes de programación. (GNU Operating System, 2015

Eclipse

Es un IDE multiplataforma y de código abierto para desarrollar clientes enriquecidos. Este entorno de desarrollo está desarrollado en Java y emplea Módulos (plug-in) para proporcionar todas las funcionalidades, por lo que puede ser fácilmente extendido a varios lenguajes de programación como C++, Python, Prolog, Perl, PHP, entre otros. Eclipse dispone de un editor de texto con: resaltado de sintaxis y completamiento de código muy potente. Además, este entorno de desarrollo pude ser integrado a sistemas de control de versiones como subversión. (Eclipse, 2014).

Qt Creator

Entorno de Desarrollo creado por Trolltech, multiplataforma, diseñado para hacer que el desarrollo en C++ de la aplicación Qt sea más rápido y fácil. Qt Creator no quiere ser un reemplazo de Eclipse ni Visual Studio, sino un IDE ligero pensado especialmente para el desarrollo en múltiples plataformas: Windows XP y Vista, Linux y Mac OSX (QT Creator, 2009).

Posee como principales características un avanzado editor de código C++, además soporta los lenguajes: C#/.NET Lenguajes (Mono), Python: PyQt , Ada, Pascal, Perl, PHP y Ruby; posee también una guía integrada y diseñador de formularios, herramientas para proyectos y administración, ayuda sensible al contexto integrada, depurador visual, resaltado y auto-completado de código y soporte para refactorización de código. (QT Digia, 2012) .

Para el desarrollo de la solución se decide utilizar el entorno integrado de desarrollo Qt Creator ya que el mismo presenta una gran integración con el framework Qt. Este IDE brinda completamiento de código para las clases pertenecientes a la biblioteca Qt y también dispone de un sistema de ayuda contextualizada.

1.8 Bibliotecas Gráficas

Las bibliotecas gráficas son colecciones de clases, funciones y otros elementos de software que brindan interfaces de alto nivel de abstracción para la visualización de información en

aplicaciones informáticas. Las bibliotecas gráficas incrementan la productividad del programador que la utiliza, debido a que este no tiene que usar instrucciones de bajo nivel para visualizar datos. En la actualidad existen una gran variedad de bibliotecas gráficas disponibles, por lo que es necesario analizar sus características antes de elegir cuál utilizar.

Qwt es una biblioteca de código abierto y multiplataforma que extiende las funcionalidades que provee el módulo QtGui del framework Qt. Esta biblioteca provee un conjunto de componentes gráficos así como clases técnicas que se utilizan para crear aplicaciones que requieren la visualización de información en forma de gráficos. Qwt también proporciona diferentes tipos de componentes para interactuar con la aplicación como: escalas, termómetros y ruedas. (Qwt SouceForge, 2008).

1.9 Biblioteca QtWebsocket

QtWebsocket es una biblioteca fundamental en el desarrollo de aplicaciones que requieren conectividad en tiempo real entre servidores y clientes, independientemente de la plataforma. Permite establecer mecanismos de sincronización en sistemas que requieran acceso exclusivo a recursos, resolviendo los principales problemas de la concurrencia en un esquema de tiempo real (Luis Vivas, 2014)

1.10 Lenguajes de programación

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos de hardware y software existentes (MARIN, 2010)

C++

C++ es un lenguaje imperativo, orientado a objetos, derivado de C. Al igual que C; C++ está muy ligado al hardware subyacente, manteniendo una considerable potencia para la programación a bajo nivel; pero se le han añadido elementos que permiten también un estilo de programación con alto nivel de abstracción. (Stroustrup, 2013)

El lenguajeC++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. (Jalon, 2009)

C++ brinda la posibilidad de crear clases, plantillas, sistema de espacios de nombres y funciones en línea, posee un mecanismo para el manejo de excepciones, permite la sobrecarga de operadores y utiliza operadores para el manejo de memoria. Este lenguaje de programación esta estandarizado por la Organización Internacional de Estándares (ISO) y cuenta con una biblioteca estándar de alta calidad. (Stroustrup, 2013)

XML

XML es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Deriva del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información. (Floyd, 2000)

XML es un lenguaje sencillo que tiene a su alrededor otras que la complementan y la hacen más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. (Floyd,2000).

1.11 Lenguaje Unificado de Modelado (UML)

UML es un lenguaje estándar de modelado visual que se usa para especificar, construir, documentar y visualizar artefactos de un sistema de software. UML permite a los desarrolladores visualizar el resultado de su trabajo en esquemas o diagramas estandarizados. Por ejemplo, símbolos o iconos característicos utilizados para capturar los requisitos. Estos iconos no son más que una notación gráfica, es decir, una síntesis; sin embargo detrás de esta notación gráfica, UML especifica un significado, es decir, una semántica. (Pressman, 2007)

Dicho lenguaje proporciona un vocabulario que incluye tres categorías: elementos, relaciones y diagramas, conteniendo aspectos conceptuales tales como procesos de negocios, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. (Pressman, 2007)

1.12 Herramienta CASE (Visual Paradigm)

Herramienta CASE (Computer Aided Software Engineering) propicia un conjunto de ayudas para

el desarrollo de programas informáticos dando soporte al modelado visual con UML (Unified Modeling Language), desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. (Lopez, 2013)

Visual Paradigm es una herramienta CASE (Ingeniería de Software Asistida por Ordenador, en inglés Computer Aided Software Engineering) que utiliza UML como lenguaje de modelado. Soporta el ciclo de vida completo de desarrollo de un software, desde la fase de análisis hasta el despliegue del mismo. Permite realizar ingeniería directa o inversa sobre el software, es capaz, a partir de un modelo relacional en diferentes SGBD, desplegar todas las clases asociadas a las tablas y soporta múltiples usuarios trabajando sobre el mismo proyecto. Visual Paradigm es una herramienta libre utilizada para el modelado de aplicaciones. (International, Visual Paradigm, 2008).

Esta herramienta es fácil de utilizar y permite construir la aplicación con mayor rapidez, mayor trabajo en equipo y mayor exactitud, facilitando la interoperabilidad con otras herramientas CASE, la mayoría de los IDEs principalmente, permitiendo así la integración de todos los componentes. (Lopez, 2013)

1.13 Metodologías de desarrollo de software

Una metodología es aquella guía que se sigue a fin de realizar las acciones propias de una investigación. En términos más sencillos se trata de la guía que nos va indicando qué hacer y cómo actuar cuando se quiere obtener algún tipo de investigación. (Mis Respuestas, 2006)

Es posible definir una metodología como aquel enfoque que permite observar un problema de una forma total, sistemática y disciplinada. El ciclo de vida del software es complejo, de ahí la necesidad de utilizar metodologías diferentes de acuerdo con las características del proyecto en desarrollo. (Mis Respuestas, 2006)

1.13.1 AUP variación UCI

El Proceso Unificado Ágil de Scott Ambler o *Agile Unified Process* (AUP) en inglés es una versión simplificada del Proceso Unificado de Rational (RUP).Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

La metodología de desarrollo AUP aplica técnicas ágiles incluyendo:

- Desarrollo dirigido por pruebas (*test driven development-TDD* en inglés)
- Modelado Ágil
- Gestión de Cambios
- Refactorización de Base de Datos para mejorar la productividad

Al no existir una metodología de software universal ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, y recursos.) exigiéndose así que el proceso sea configurable, se decide hacer una variación de la metodología AUP de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. Una metodología de desarrollo de software tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas y para ello, nos apoyaremos en el Modelo CMMI-DEV v1.3, el cual constituye una guía para aplicar las mejores prácticas en una entidad desarrolladora. Estas prácticas se centran en el desarrollo de productos y servicios de calidad.

Fases

- **Inicio**: En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
- **Ejecución**: En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del software.
- **Cierre**: En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

1.14 Conclusiones Parciales

En este capítulo se realizó un esbozo de las principales características de los sistemas de simulación de datos y se logró tomar experiencias del comportamiento y las funcionalidades que ofrecen algunos de los sistemas de simulación usados en el mundo, que servirán como base para comenzar el desarrollo de la aplicación enfocada además en las necesidades propias del sistema. Luego del análisis de las tecnologías, lenguajes y bibliotecas posibles a utilizar en el desarrollo de la solución se está en condiciones, de acuerdo a sus beneficios o dificultades de uso de cada una de ellas, decidir cuál es la más adecuada para el desarrollo de la aplicación propuesta, lo cual quedó definida de la siguiente forma:

- Plataforma de desarrollo: **Debian 7 (Wheezy)**
- Framework de desarrollo: **Framework QT**
- Lenguaje de programación: **C++ , XML**
- Entorno de desarrollo: **Qt Creator**
- Biblioteca gráfica**: Qwt**
- Biblioteca: **QtWebsocket**
- Herramienta case : **Visual Paradigm**
- Lenguaje de Modelado: **UML**
- Metodología de desarrollo: **AUP variación UCI**

Capítulo 2
Análisis y diseño de la herramienta.

2.1 Introducción

El presente capítulo tiene como objetivo reflejar las actividades realizadas en los procesos de análisis y diseño de la herramienta de simulación. En el mismo se exponen los artefactos más importantes que describen el flujo normal de eventos que ocurren en el sistema, tales como especificaciones de requisitos que rigen el desarrollo de la solución, diagramas y las principales clases que componen la herramienta, detallando la información del análisis y del diseño de la solución en cuestión.

2.2 Modelo de dominio

Un modelo conceptual o modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las cosas que existen o los eventos que suceden en el entorno en el que trabaja el sistema. El objetivo del modelo de dominio es comprender y describir las clases más importantes dentro del contexto del sistema y así contribuir a una comprensión del problema. (JACOBSON, 2000)

2.2.1 Diagrama de clases del modelo de dominio

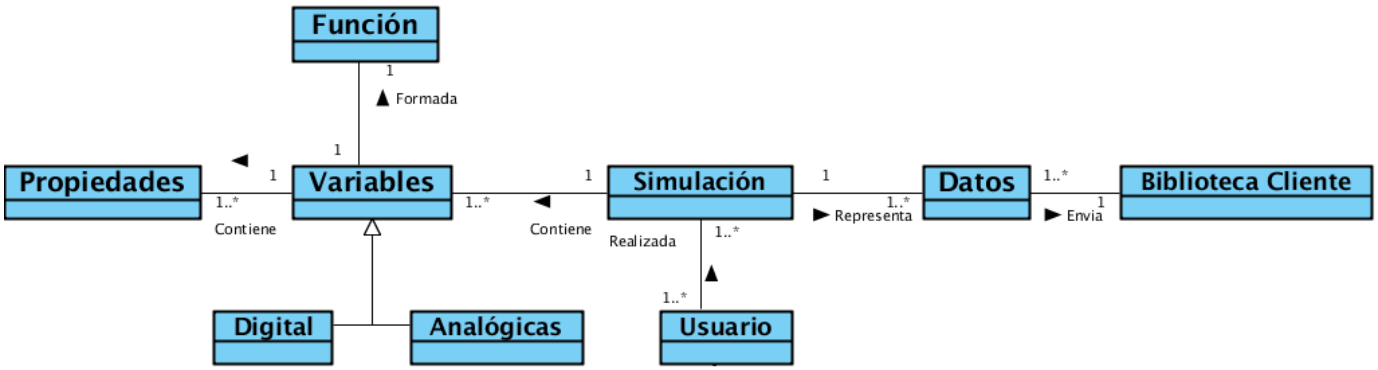

Figura 5: Modelo de Dominio

2.2.2 Descripción del modelo de dominio

El sistema propuesto para satisfacer el problema a resolver es una herramienta que permite crear variables de 2 tipos, digitales y analógicas cada una con sus propiedades. Una variable está formada por una función de simulación que a su vez también presenta diferentes

propiedades de acuerdo a su método de simulación. La simulación se lleva a cabo con esas variables representando datos y dicha simulación es realizada por un usuario del SCADA Guardián del Alba. Los datos son enviados a través de una librería o biblioteca hacia una interfaz cliente.

2.3 Actores del sistema

Se define como persona relacionada con el sistema, a aquella que interactúa de una forma u otra con la herramienta propuesta , dígase vinculada al proceso de desarrollo así como a las que interactúa con el sistema a realizar. (Pressman, 2007)

Tabla 1: Actor de sistema

Actor	Descripción
Usuario	Actor del negocio encargado de gestionar las funcionalidades del sistema.

2.4 Requisitos funcionales y no funcionales del sistema.

En este acápite se muestran en forma de listado los distintos requisitos funcionales y no funcionales del software. Los requisitos funcionales son condiciones o capacidades que el software debe cumplir y los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable (PRESSMAN, 2010).

2.4.1 Requisitos funcionales

RF1 Cargar configuración XML: Esta funcionalidad permite cargar en formato XML las variables asignadas a funciones y propiedades establecidas.

RF2 Salvar configuración XML: Esta funcionalidad permite salvar en formato XML variables asignadas a funciones con sus propiedades establecidas.

RF3 Crear variables analógicas: Esta funcionalidad permite crear variables de tipo analógicas.

RF4 Crear variables digitales: Esta funcionalidad permite crear variables de tipo digitales.

RF5 Modificar una variable analógica: Esta funcionalidad permite modificar las propiedades de las variables de tipo analógicas previamente creadas en el sistema.

RF6 Modificar una variable digital: Esta funcionalidad permite modificar las propiedades de las variables de tipo digital previamente creadas en el sistema.

RF7 Eliminar variable analógica: Esta funcionalidad permite eliminar las variables de tipo analógicas creadas en el sistema.

RF8 Eliminar variable digital: Esta funcionalidad permite eliminar las variables de tipo digital creadas en el sistema.

RF9 Crear una función: Esta funcionalidad permite crear las funciones de simulación con sus propiedades establecidas.

RF10 Modificar una función: Esta funcionalidad permite modificar las propiedades de las funciones asignadas a las variables en el sistema.

RF11 Eliminar una función: Esta funcionalidad permite eliminar las funciones existentes en el sistema.

RF12 Ejecutar simulaciones: Esta funcionalidad permite que se realicen ejecuciones de simulaciones de datos.

RF13 Graficar las variables: Esta funcionalidad permite que se visualice el comportamiento de los valores de las variables perteneciente a las simulaciones ejecutándose en el sistema.

RF14 Enviar los datos simulados a la aplicación cliente: Esta funcionalidad permite que los valores sean enviados a través de una biblioteca hacia una interfaz cliente.

2.4.2 Requisitos no funcionales

Requisitos no funcionales de usabilidad:

RNF1: El sistema debe ser usado por cualquier persona que tenga conocimientos básicos de computación.

RNF2: La información deberá estar disponible en todo momento, limitada solamente por las restricciones que se tengan de acuerdo a las políticas de seguridad del sistema.

RNF3: Se debe lograr un producto configurable y extensible, de manera que sea posible incorporar nuevas funcionalidades, sin que se afecte el mismo.

RNF4: La aplicación contará con íconos representativos.

Requisitos no funcionales de interfaz de usuario:

RNF5: La herramienta debe tener una interfaz gráfica uniforme incluyendo pantallas, menús y opciones.

RNF6: La consistencia de la interacción entre usuario y componente estará determinada por el diseño de la interfaces de usuario que mantendrán los elementos como menús y zonas de trabajo, en posiciones fijas, además de la mayor uniformidad posible entre cuadros de texto y botones.

RNF7: Tanto los títulos de los componentes de la interfaz, como los mensajes para interactuar con los usuarios, así como los mensajes de error, deberán tener una apariencia uniforme. Los mensajes de error deberán ser lo suficientemente informativos para dar a conocer la severidad del error.

RNF8: Se deberá facilitar la entrada de datos a los usuarios, presentando campos de selección que permitan escoger los valores posibles con los que se podrá llenar un determinado elemento en la interfaz, haciendo que el proceso de llenado de datos sea lo más intuitivo posible, de tal forma que los usuarios se puedan adaptar fácilmente.

Requisitos no funcionales de hardware:

RNF9: Las computadoras que utilizarán el software a desarrollar deberán tener 2 GB de Memoria RAM como mínimo y un procesador Intel Core i3 multinúcleo con velocidad igual o superior a 2.0 GHz.

RNF10: Las computadores deben tener 1 tarjeta de red PCIe Ethernet Gigabit de 10/100/1000 Mbps.

Requisitos no funcionales de portabilidad:

RNF11: El sistema operativo será GNU/Linux con la distribución Debian Wheezy

2.5 Diagrama de casos de uso del sistema

Un diagrama de casos de uso del sistema describe parte del modelo de casos de uso y muestra un conjunto de casos de uso y actores con una asociación entre cada par actor/caso de uso que interactúan. (Jacobson, y otros, 2004).

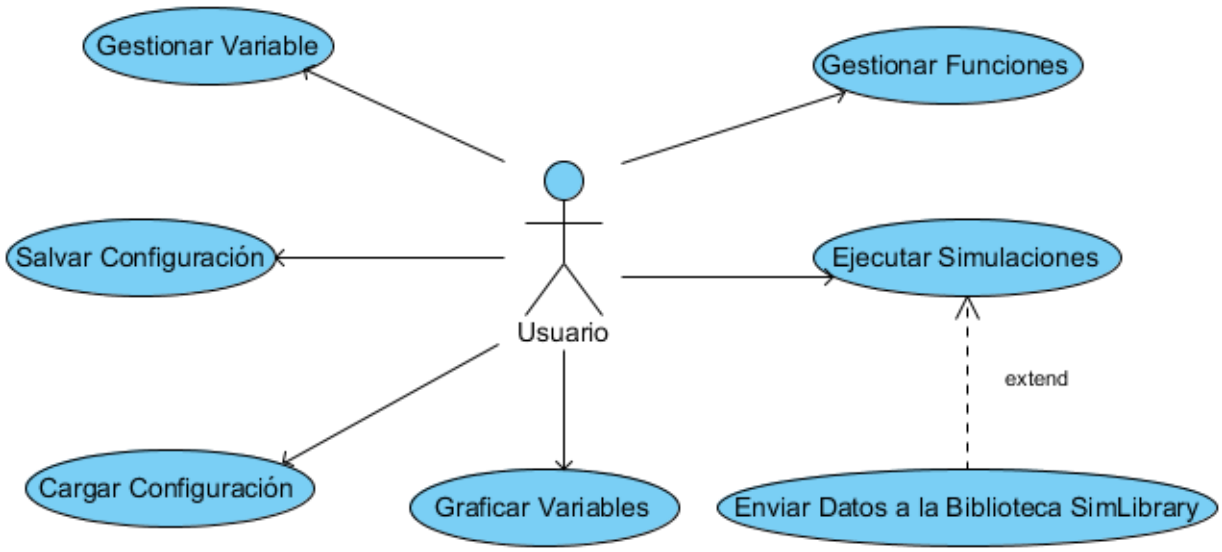

Figura 6: Diagrama de casos de uso del sistema.

2.5.1 Descripción textual de los casos de uso del sistema

Un caso de uso es una unidad coherente de funcionalidad, externamente visible, proporcionada por una unidad del sistema y expresada por secuencias de mensajes intercambiados por la unidad del sistema y uno o más actores.

El propósito de un caso de uso es definir una pieza de comportamiento coherente, sin revelar la estructura interna del sistema. Desde el punto de vista de los usuarios, éstas pueden ser situaciones anormales. Desde el punto de vista de los sistemas, son las variaciones adicionales que deben ser descritas y manejadas. (Rumbaugh, y otros, 2005).

Tabla 2: Caso de uso Cargar Configuración

Caso de Uso	Cargar Configuración
Actores	Usuario
Resumen	El caso de uso se inicia cuando un usuario del sistema decide importar en formato XML las variables y funciones con sus propiedades.
Referencia	RF1
Complejidad	Media.

Prioridad	Crítico.
Precondiciones	Debe existir almacenada una configuración en formato XML.
Poscondiciones	Importa de manera exitosa la configuración.

Flujo de eventos		
Flujo básico		
	Actor	**Sistema**
1.	**1.1** El usuario selecciona **Cargar SIM** del menú superior.	**1.2** El sistema muestra una ventana dando la posibilidad de buscar el archivo de origen que se desea cargar.
2.	**2.1** El usuario importa el archivo deseado.	**2.2** El sistema muestra en la aplicación todas las variables cargadas. **2.3** Culmina el Caso de Uso.
Prototipo de interfaz de Usuario		

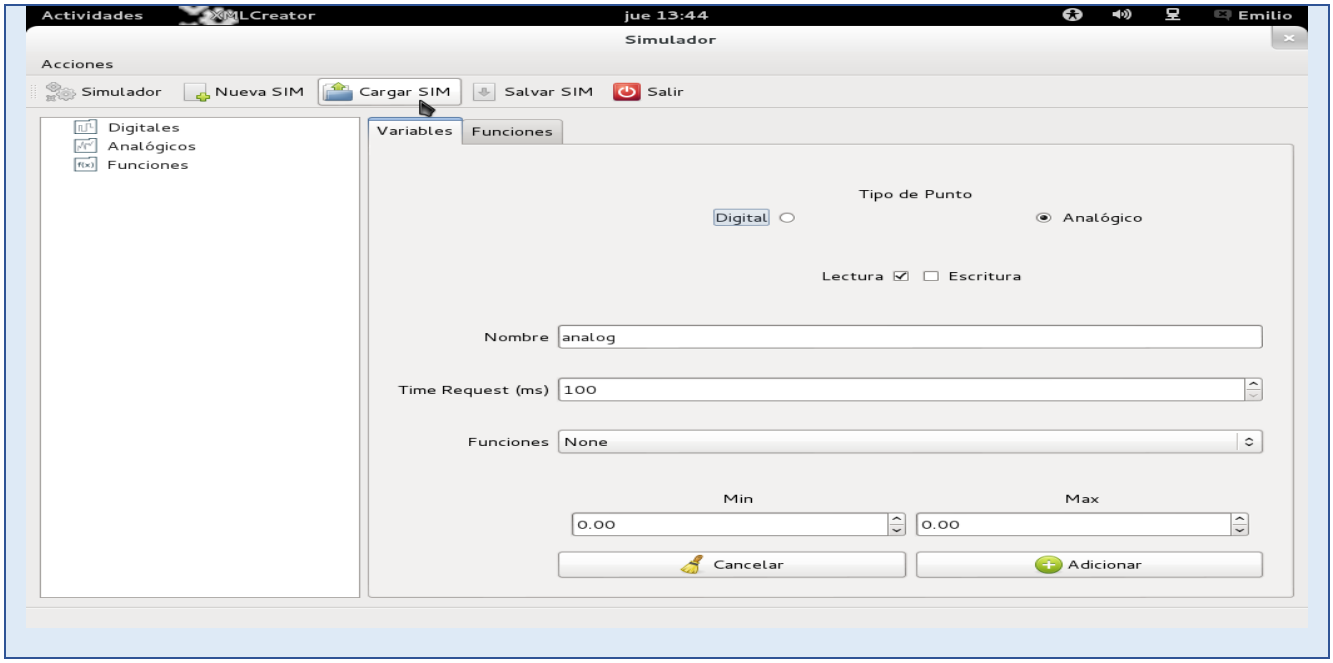

Tabla 3: Caso de uso Salvar Configuración.

Caso de Uso	Salvar Configuración
Actores	Usuario
Resumen	El caso de uso se inicia cuando un usuario del sistema decide salvar en formato XML las variables y funciones creadas con sus propiedades.
Referencia	RF2
Complejidad	Media.
Prioridad	Crítico.
Precondiciones	Deben estar creados en el sistema los datos.
Poscondiciones	Exporta de manera exitosa la configuración.

Flujo de eventos		
Flujo Básico		
	Actor	**Sistema**
1	**1.1** El usuario escoge la opción **Salvar SIM** del menú superior en la interfaz de la aplicación.	**1.2** El sistema muestra una ventana dando la posibilidad de guardar el archivo en un destino deseado.
2	**2.1** El usuario guarda el archivo en el destino deseado. **2.2** Culmina el Caso de Uso.	

Prototipo de interfaz de Usuario

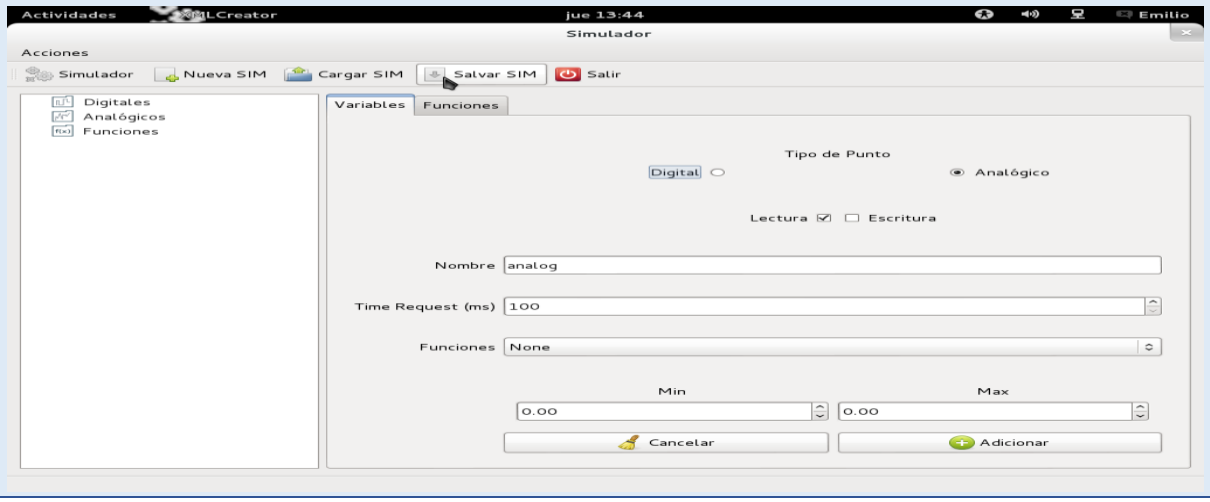

Tabla 4: Caso de uso Gestionar Variable

Caso de Uso	Gestionar Variable
Actores	Usuario

Resumen	El caso de uso se inicia cuando un usuario del sistema decide crear, modificar o eliminar una variable de tipo analógica o digital en el sistema.
Referencia	RF3, RF4, RF5, RF6, RF7, RF8
Complejidad	Media.
Prioridad	Crítico.
Precondiciones	-
Poscondiciones	-

Crear Variable		
Flujo Básico Crear Variable		
	Actor	**Sistema**
	1.1 El usuario escoge la opción **Variables** en la parte derecha de la interfaz de la aplicación.	**1.2** El sistema muestra los campos:**Digital,Analógico,Lectura,Escritura Nombre, Tiempo de Respuesta,Dirección, Funciones, Min, Max.**
	2.1 El usuario llena los campos:**Digital,Analógico,Lectura, Escritura Nombre, Tiempo de Respuesta,Dirección, Funciones, Min, Max**. **2.2** El usuario presiona el botón **Adicionar** ubicado en la parte inferior de la interfaz de la aplicación.	**2.3** El sistema valida que no existan campos vacíos(Alterno1). **2.4** El sistema valida que no exista esa variable. (Alterno 2). **2.5** El sistema muestra la variable creada en el árbol que se muestra en la parte izquierda de la interfaz de la aplicación. **2.6** Culmina el Caso de Uso.

Modificar Variable	
Flujo Básico Modificar Variable	
Actor	**Sistema**
1.1 El usuario selecciona del árbol que aparece en la parte izquierda de la interfaz de la aplicación la variable de tipo **Analógica** o **Digital** que desea modificar.	**1.2** El sistema visualiza la variable escogida por el usuario mostrando los datos de los campos:**Digital,Analógico,Lectura,Escritura Nombre, Tiempo de Respuesta,Dirección, Funciones, Min, Max.**
2.1 El usuario modifica el campo deseado. **2.2** Presiona el botón **Actualizar** de la parte inferior.	**2.3** Verifica que los campos no estén vacíos (Alterno 1). **2.4** El sistema actualiza la variable con sus nuevos datos. **2.5** Culmina el Caso de Uso.
Eliminar Variable	
Flujo Básico Eliminar Variable	
Actor	**Sistema**
1.1 El usuario selecciona del árbol creado en la parte izquierda de la interfaz de la aplicación la variable de tipo **Analógica o Digital** anteriormente se creó. **1.2** El usuario presiona el clic derecho sobre la variable escogida a eliminar y presiona la opción **Eliminar**.	**1.3** El sistema muestra un mensaje de confirmación **"Está seguro de eliminar la variable X?"**

	2.1 El usuario escoge la opción **Si.**	**2.2** El sistema muestra un mensaje **"Variable Eliminada"**
Flujo Alterno de Eventos 1 Campos vacíos		
	Actor	**Sistema**
		1.1 El sistema muestra el mensaje **"Campos vacíos".** **1.2** Regresa al paso 2.1 del flujo básico.
Flujo Alterno de Eventos 2 Variable Existente		
	Actor	**Sistema**
		1.1 El sistema muestra el mensaje **"Variable Existente"** **1.2** Regresa al paso 2.1 del flujo básico**.**
Prototipo de Interfaz de usuario		

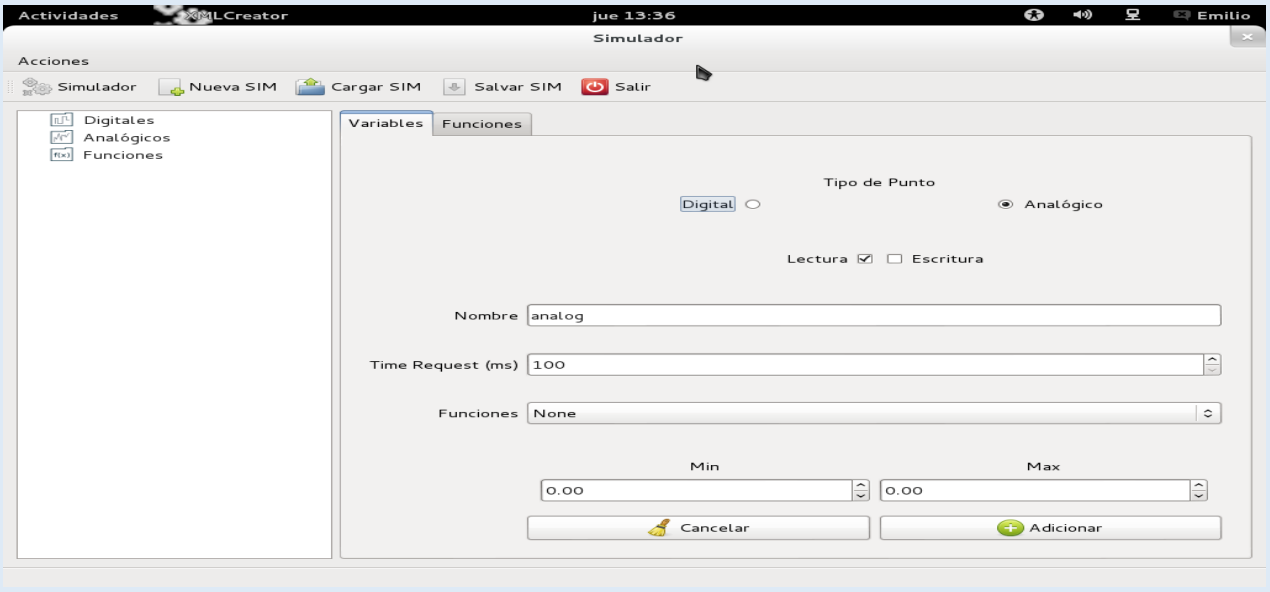

Tabla 5: Caso de uso Gestionar Función.

Caso de Uso	Gestionar Función	
Actores	Usuario	
Resumen	El caso de uso se inicia cuando el usuario del sistema decide crear, modificar o eliminar una función de tipo **IncDouble, IncInt, DecDouble, DecInt, SinDouble, CosDouble, RampDouble, RampInt, RandDouble, RandInt.**	
Referencia	RF9, RF10, RF11	
Complejidad	Media.	
Prioridad	Crítico.	
Precondiciones	-	
Poscondiciones	Se crea la función de simulación.	
Crear Función		
Flujo básico Crear Función.		
	Actor	**Sistema**
1	**1.1** El usuario escoge la opción **Funciones**	**1.2** El sistema muestra los campos**: Nombre, Tipo,Periodo,Min,Inc,Max.**
2	**2.1** El usuario introduce los datos. **2.2** El usuario presiona el botón **Adicionar** situado en la parte inferior de la interfaz de la aplicación .	**2.3** El sistema valida que los campos no estén vacíos (Alterno 1). **2.4** El sistema valida que las función no exista (Alterno 2). **2.5** El sistema muestra la función creada en el

		árbol que se muestra en la parte izquierda. **2.6** Culmina el Caso de Uso.
Modificar Función		
Flujo Básico Modificar Función		
	Actor	**Sistema**
1	**1.1** El usuario selecciona en el árbol ubicado en la parte izquierda de la interfaz de la aplicación la función que quiere modificar.	**1.2** El sistema muestra la función escogida por el usuario y muestra los datos de los campos: **Nombre, Tipo, Periodo,Min,Inc,Max.**
2	**2.1** El usuario modifica el campo que desea modificar. **2.2** Presiona el botón **Actualizar** de la parte inferior.	**2.3** Verifica que los campos no estén vacíos (**Alterno 1**). **2.4** El sistema actualiza nuevamente la función con sus nuevos datos.
Eliminar Función		
Flujo Básico Eliminar Función		
	Actor	**Sistema**
1	**1.1** El usuario escoge del árbol ubicado en la parte izquierda de la interfaz de la aplicación la función que quiere eliminar dando **clic** sobre ella. **1.3** El usuario escoge la opción **Si.**	**1.2** El sistema muestra un mensaje de confirmación **"¿Está seguro de Eliminar?".** **1.4** El sistema muestra un mensaje **''Función Eliminada".**
Flujo Alterno 1 Campos Vacíos		
	Actor	**Sistema**

1		**1.1** El sistema muestra el mensaje **"Campos vacíos".** **1.2** Regresa al paso 1.2 del flujo básico.

Flujo Alterno 2 Función Existente		
	Actor	**Sistema**
1		**1.1** El sistema muestra el mensaje **"Función Existente".** **1.2** Regresa al paso 1.2 del flujo básico.

Prototipo de interfaz de Usuario

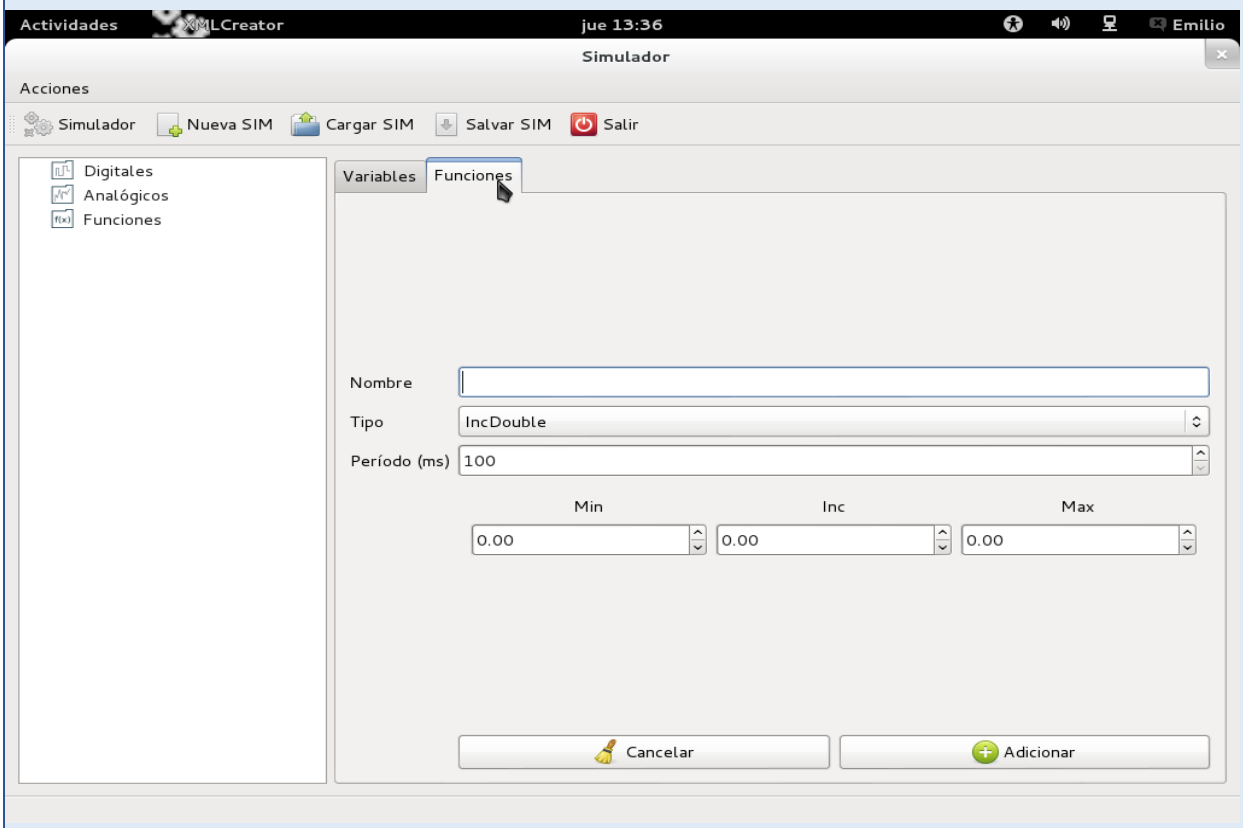

Tabla 6: Caso de uso Ejecutar Simulaciones

Caso de Uso		Ejecutar Simulaciones
Actores		Usuario
Resumen		El caso de uso se inicia cuando un usuario del sistema decide realizar el proceso de simulación.
Referencia		RF12
Complejidad		Media.
Prioridad		Crítico.
Precondiciones		Tiene que existir variables asignadas a funciones creadas para poder simular datos.
Poscondiciones		Simula los datos.
Flujo de eventos		
Flujo básico		
	Actor	**Sistema**
1	**1.1** El usuario selecciona la opción **Iniciar Simulación** ubicado en la parte superior de la interfaz de la aplicación**.**	**1.2** El sistema muestra una lista de todas las variables con sus funciones asignadas.
2	**2.1** El usuario presiona el botón **Iniciar.**	**2.2** El sistema muestra en la aplicación todos los valores dados por las variables existentes. **2.3** Culmina el Caso de Uso.

Prototipo de interfaz de usuario

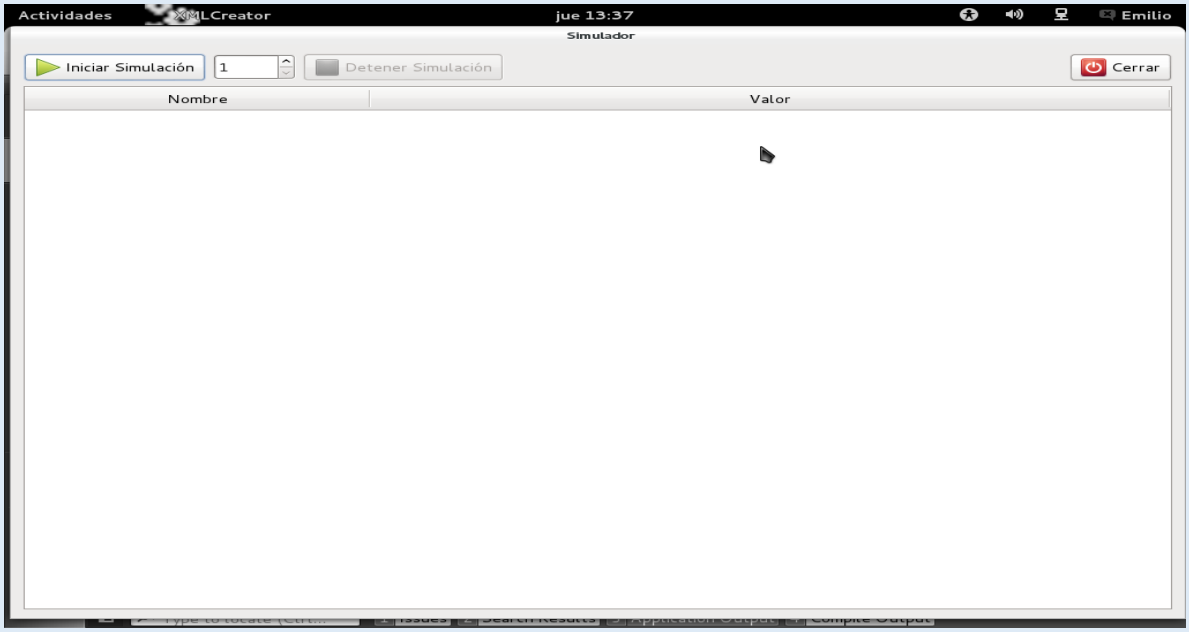

Tabla 7: Caso de uso Graficar

Caso de Uso	Graficar Variable
Actores	Usuario
Resumen	El caso de uso se inicia cuando un usuario del sistema decide realizar el proceso de visualización de una variable.
Referencia	RF13
Complejidad	Media.
Prioridad	Crítico.
Precondiciones	Simulaciones existentes en la herramienta
Poscondiciones	Grafica la variable escogida
Flujo de eventos	

Flujo básico		
	Actor	**Sistema**
1	**1.1** El usuario selecciona la opción **Iniciar Simulación** ubicado en la parte superior de la interfaz.	**1.2** El sistema muestra una lista de todas las variables con sus funciones asignadas.
2	**2.1** El usuario presiona el botón **Iniciar**.	**2.2** El sistema muestra en la aplicación todos los valores dados por las variables existentes.
3	**3.1** Se escoge la variable que se quiera ver el comportamiento de su visualización.	**3.2** El sistema muestra una gráfica en 2 dimensiones su comportamiento. **3.3** Culmina el Caso de Uso.

Prototipo de interfaz de usuario

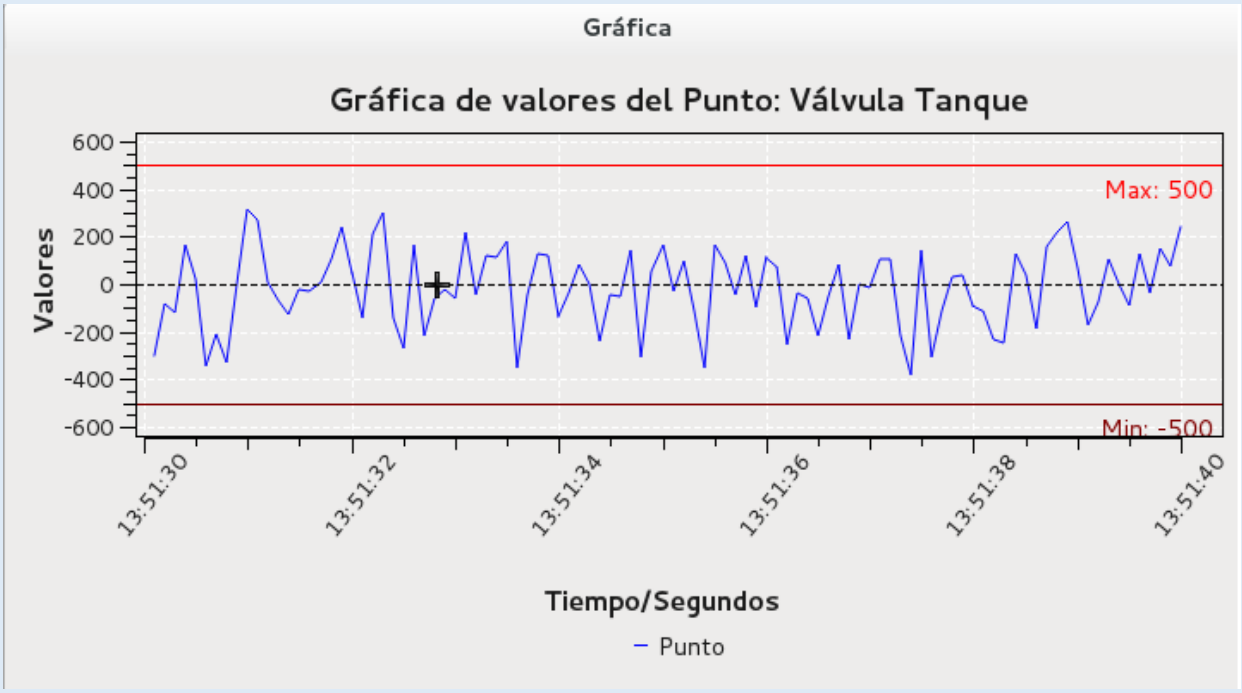

Tabla 7: Caso de uso Enviar datos a la Biblioteca SimLibrary

Caso de Uso		**Enviar datos a la Biblioteca SimLibrary**
Actores		Usuario
Resumen		El caso de uso se inicia cuando un usuario del sistema decide realizar el proceso de simulación y enviar los datos a la biblioteca SimLibrary.
Referencia		RF14
Complejidad		Media.
Prioridad		Crítico.
Precondiciones		Simulaciones existentes en la herramienta.
Poscondiciones		-Envía los datos a la biblioteca.
Flujo de eventos		
Flujo básico		
	Actor	**Sistema**
1	**1.1** El usuario selecciona la opción **Iniciar Simulación**.	**1.2** El sistema muestra una lista de todas las variables con sus funciones asignadas.
2	**2.1** El usuario presiona el botón **Iniciar**.	**2.1** El sistema muestra en la aplicación todos los valores dados por las variables existentes.

2.6 Arquitectura del sistema

Según el Instituto de Ingeniería Eléctrica y Electrónica (IEEE), la arquitectura del software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre

ellos y el contexto en el que se implantarán y los principios que orientan su diseño.

Para la solución propuesta se definió utilizar el patrón arquitectónico Modelo- Vista- Controlador (MVC) y la arquitectura Cliente-Servidor.

El patrón Modelo-Vista-Controlador separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones, es decir, se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas. (Alvarez, 2014).

De manera genérica, los componentes de MVC (Alvarez, 2014) se podrían definir como sigue:

- **El Modelo:** Es la representación de la información con la cual el sistema opera, por lo tanto, gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

- **El Controlador:** Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto, se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'

- **La Vista:** Presenta el 'modelo' en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto, requiere de dicho 'modelo' la información que debe representar como salida.

La aplicación del patrón MVC en la creación de un software brinda a sus desarrolladores un número de beneficios que permiten agilizar la implementación, dentro de ellos se pueden encontrar:

- Facilita y flexibiliza la estructuración del código.

- Clara separación de responsabilidades entre interfaz, lógica de negocio y de control.
- Permite la reutilización de los componentes.
- Tiene facilidad para desarrollar prototipos rápidos

A continuación se muestra un ejemplo del **Modelo-Vista–Controlador** utilizado en la solución propuesta:

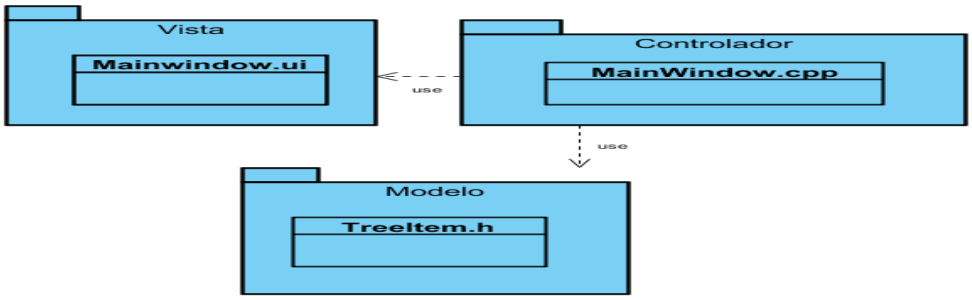

Figura 7: Ejemplo del Uso de la Arquitectura MVC

- **MainWindow.ui:** Vista en la cual el usuario interactúa con todos las funcionalidades de la solución propuesta.
- **MainWindow.cpp:** Es la controladora encargada de manejar los operaciones de las variables digitales con sus funciones asignadas.
- **TreeItem:** Es el modelo que maneja la creación de variables digitales y analógicas.

Con respecto a la definición de arquitectura **Cliente-Servidor** se encuentran las siguientes definiciones:

- Cualquier combinación de sistemas que pueden colaborar entre sí para dar a los usuarios toda la información que ellos necesiten sin que tengan que saber dónde está ubicada.
- Es una arquitectura de procesamientos cooperativa donde uno de los componentes pide servicios a otro.
- El término cliente/servidor es originalmente aplicado a la arquitectura de software que describe el procesamiento entre dos o más programas: una aplicación y un servicio soportante.
- El modelo soporta un medio ambiente distribuido en el cual los requerimientos de servicio hechos por estaciones de trabajo inteligentes o "clientes", resultan en un trabajo realizado por otros computadores llamados servidores".

La arquitectura **Cliente-Servidor** contempla las siguientes características (Valle, 2009):

- El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de Cliente y Servidor pueden estar en plataformas separadas o en la misma plataforma.
- Un servidor da servicio a múltiples clientes en forma concurrente.
- La interrelación entre el hardware y el software están basados en una infraestructura poderosa, de tal forma que, el acceso a los recursos de la red no muestra la complejidad de los diferentes tipos de formatos de datos y de los protocolos.
- Su capacidad para permitir integrar los equipos ya existentes en una organización, dentro de una arquitectura informática descentralizada y heterogénea.

En conclusión, Cliente/Servidor puede incluir múltiples plataformas, bases de datos, redes y sistemas operativos. Estos pueden ser de distintos proveedores, en arquitecturas propietarias y no propietarias y funcionando todos al mismo tiempo.Por lo tanto, su implantación involucra diferentes tipos de estándares, ejecutándose sobre diversos sistemas operativos, utilizando medios de comunicación heterogéneos, solo por mencionar algunas de las posibilidades (Valle, 2009).

A continuación un ejemplo de cómo se usó la arquitectura **Cliente-Servidor** en la solución propuesta:

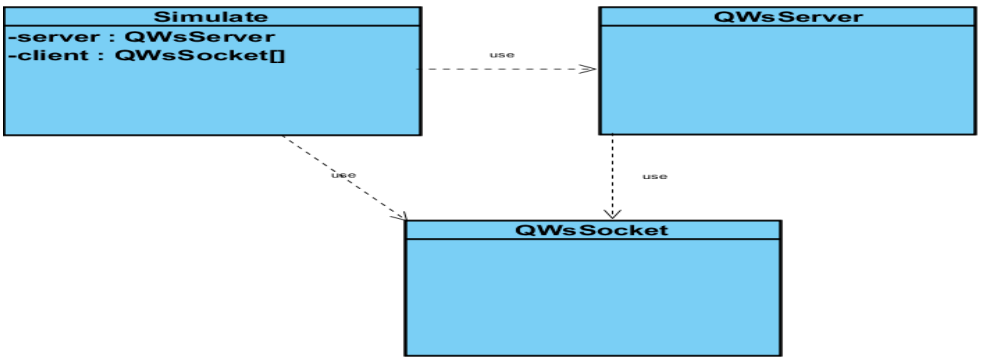

Figura 8: Uso de la arquitectura Cliente/Servidor

- **QwsSocket:** Clase encargada de enviar y recibir los datos además de establecer la comunicación.
- **QwsServer:** Clase encargada de recibir los datos y crea un objeto socket estableciendo

una comunicación con la clase **QwsSocket.**

2.7 Patrones de diseño

Un patrón de diseño es una semilla de conocimiento, que tiene un nombre y transporta la esencia de una solución probada a un problema recurrente dentro de cierto contexto en medio de intereses en competencia. Dicho de otro modo, un patrón de diseño describe una estructura que resuelve un problema particular dentro de un contexto específico y en medio de fuerzas que pueden tener un impacto en la manera en que se aplica y utiliza el patrón. Se deben tener presentes los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios). (Pressman, 2007)

Los patrones de diseño reflejan todo el rediseño y remodificación que los desarrolladores han ido haciendo a medida que intentaban conseguir mayor reutilización y flexibilidad en su software. Los patrones documentan y explican problemas de diseño y luego discuten una buena solución a dicho problema. Los principales patrones de diseño son los conocidos como GoF (del inglés Gang Of Four, en español grupo de los cuatro). En este grupo de patrones (UPM, 2012) se encuentran los:

- **Creacionales** (*Abstract Factory, Builder, Factory Method, Prototype, Singleton*).
- **Estructurales** *(Adapter, Bridge, Decorator, Facade*, entre otros).
- **Patrones de comportamiento** (*Command, Observer, State, Visitor*, *Template Method*, entre otros).

A continuación se muestra los patrones de diseño utilizados en la solución propuesta:

Template Method

El propósito del Template Method es definir el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, permitiendo que las subclases cambien pasos de un algoritmo sin cambiar su estructura. (UPM, 2012).

Aplicabilidad.

- Para implementar las partes de un algoritmo que no cambian y dejar que las subclases implementen el comportamiento que puede variar.

- Cuando el comportamiento repetido de varias subclases debería factorizarse y localizarse en una clase común, para evitar código duplicado.
- Para controlar las extensiones de las subclases.

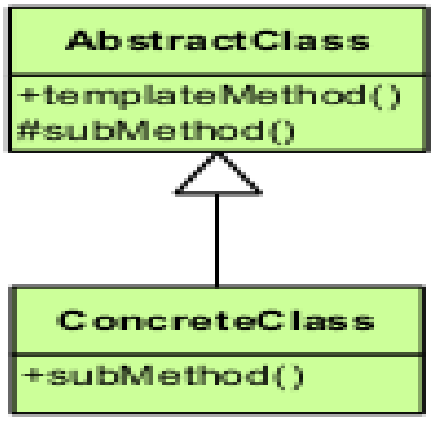

Figura 9: Estructura Patrón *Template Method*

A continuación se muestra en la figura 9 el uso del patrón *Template Method* en la solución:

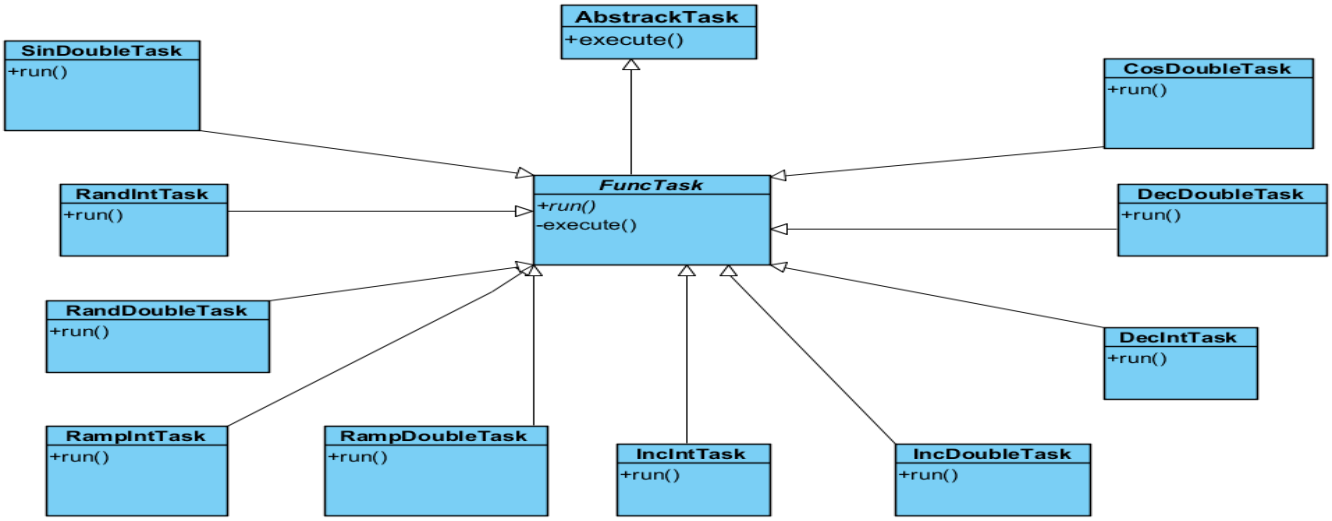

Figura 10: Uso del patrón Template Method

El uso del patrón *Template Method* se evidencia en la herramienta en que cada tarea tiene implementado de manera propia su método de funcionamiento llamado *run(),* método el cual hereda de la clase *FuncTask*, los atributos que tiene cada tarea siendo este método virtual puro.

Abstract Factory

El patrón *Abstract Factory* está aconsejado cuando se prevé la inclusión de nuevas familias de

productos pero puede resultar contraproducente cuando se añaden nuevos productos o cambian los existentes puesto que afectaría a todas las familias creadas. El problema que intenta solucionar este patrón es el de crear diferentes familias de objetos.(UPM, 2012)

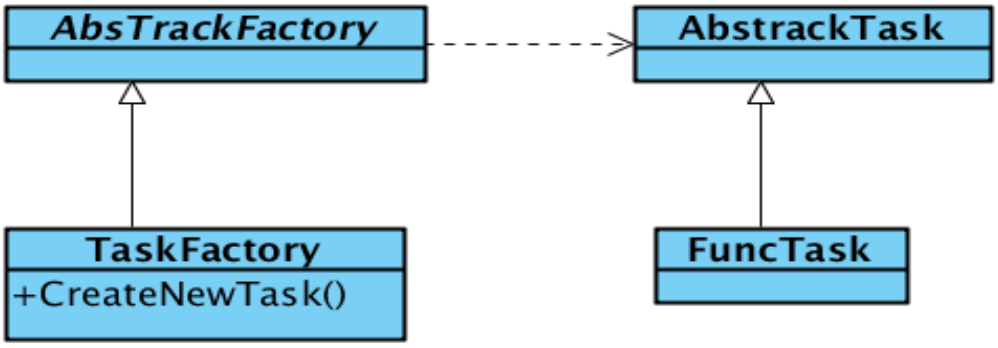

Figura 11: Uso del Patrón *Abstract Factory*

2.8 Diagrama de Clases

El diagrama de clases de diseño de un sistema refleja los detalles que tienen que ver más concretamente con la implementación, mostrando la estructura interna del sistema en cuanto a la información concerniente a cada una de las clases que forman el mismo, atributos y sus tipos de datos, métodos y sus tipos de datos de retorno y además brinda una representación gráfica de las relaciones entre todas las clases del sistema.

A continuación se muestran los diagramas de clases del diseño de los casos de usos críticos del sistema propuesto.

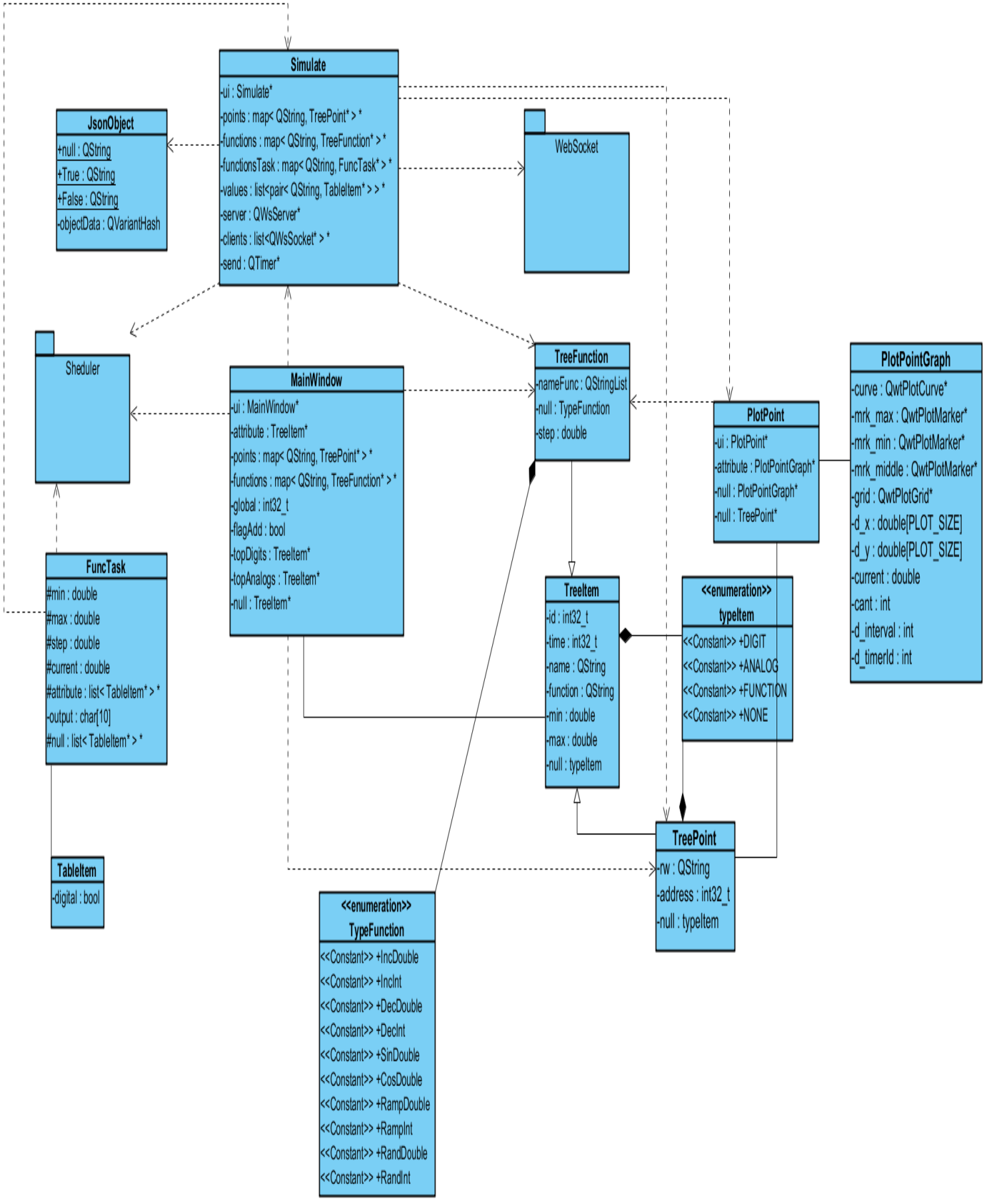

Figura 12: Diagrama de clases simulador

A continuación se muestra información acerca de las principales clases que están representadas en el diagrama de clases:

- ***Simulate*:** Clase que ejecuta las funciones y actualiza las variables analógicas y digitales asociadas a dichas funciones para la simulación de datos.
- ***PlotPoint*:** Clase que permite graficar las variables analógicos o digitales con sus funciones asociadas.
- ***FuncTask*:** Clase padre de todas las tareas.
- ***TreeFunction*:** Clase que maneja los componentes gráficos de las funciones.

El paquete ***Scheduler*** es un conjunto de clases utilizadas en el SCADA Guardián del Alba para la ejecución de las tareas concurrentemente; este paquete tiene implementada clases que facilitan el trabajo con operaciones a realizar que están en una cola con prioridad ordenada dependiendo del período de ejecución de la tarea y ejecuta concurrentemente las clases según vaya terminando con una de las tareas que tenga en ejecución.

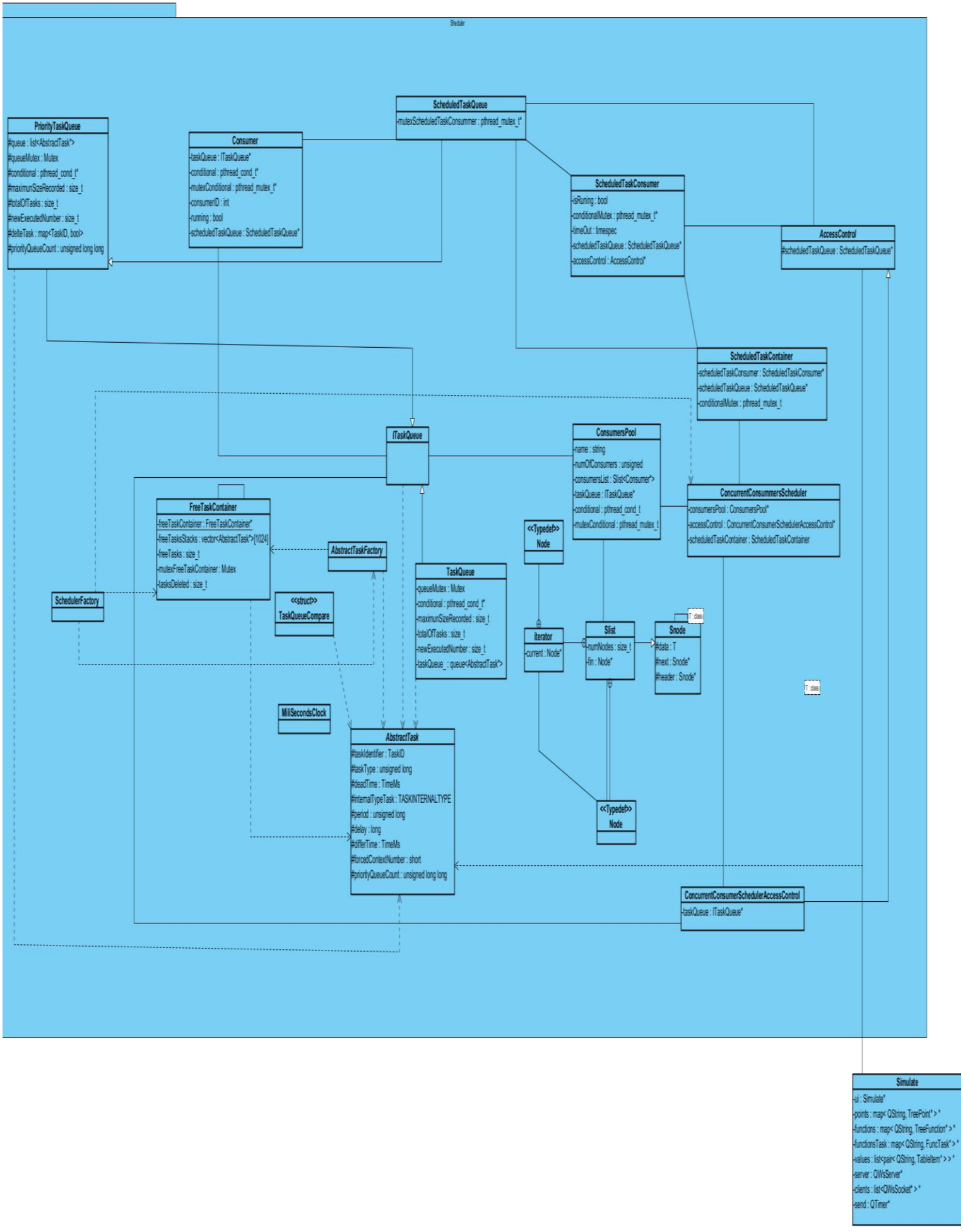

Figura 13: Diagrama de clases paquete *Scheduler*.

A continuación se muestra información acerca de las principales clases que están representadas en el diagrama de clases del paquete ***Scheduler***:

- ***AccesControl*:** Clase encargada de manejar el control de acceso a cualquier tipo de Schedule instanciado.
- ***Slist*:** Clase para el manejo de listas simplemente enlazadas.
- ***TaskQueque*:** Clase para el manejo de colas de tareas, estas colas son protegidas por un *mutex* e informan a los hilos de la existencia de una nueva tarea en la cola.
- ***Consumer*:** Clase utilizada para consumir las tareas en cola, los consumidores se pelean por consumir las tareas.
- ***Snode*:** Clase utilizada para representar un nodo con un enlace al siguiente nodo en una lista.
- ***AbstractTask*:** Clase abstracta del Scheduler para la ejecución de las tareas en el mismo.

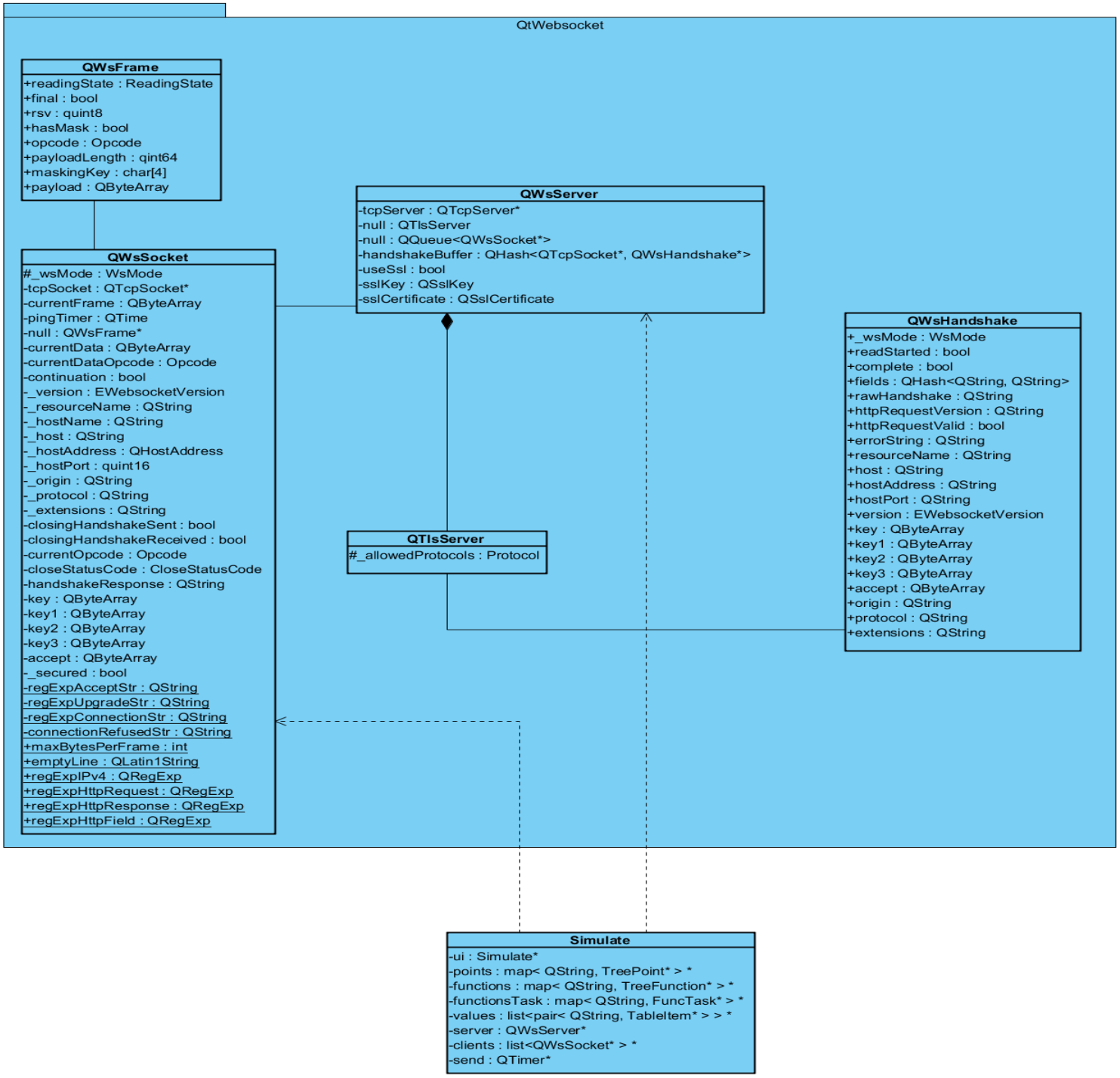

Figura 14: Diagrama de clases paquete *WebSocket*

A continuación se muestra información acerca de las principales clases que están representadas en el diagrama de clases del paquete ***WebSocket***:

- ***QwsSocket*:** Clase encargada de enviar y recibir los datos además de establecer la comunicación.
- ***QwsServer*:** Clase encargada de recibir los datos y crea un objeto socket estableciendo una comunicación con la clase ***QwsSocket*.**

2.9 Conclusiones Parciales

Siguiendo la metodología de desarrollo AUP, en este capítulo se hizo una descripción detallada de los actores del sistema, de los casos de uso del negocio y de las reglas del negocio. También se presentó el diagrama de clases según su importancia y se sintetizaron los principales requerimientos funcionales y los no funcionales con el objetivo de hacer fácil la construcción del sistema.

Con todos los elementos expuestos en este capítulo, se puede proceder a desarrollar el diseño y la construcción del sistema, vigilando que se cumplan todas las funcionalidades anteriormente analizadas.

Capítulo 3
Implementación y prueba del sistema

3.1 Introducción

El presente capítulo presenta la implementación del sistema con el objetivo de mostrar los componentes que se obtienen de implementar las clases y los paquetes definidos en el diseño. También describe la vista de implantación del sistema y los casos de pruebas desarrollados para validar la solución implementada.

3.2 Estilos de Códigos

3.2.1 Definiciones de clases

Las declaraciones de clases tienen su llave de apertura una línea más abajo de la declaración y el nombre de la clase comienza con mayúscula. Los nombres de las mismas son sustantivos singulares, que deben reflejar qué hacen y no cómo lo hacen. A continuación se muestra un ejemplo de código de definición de clase usado en la herramienta.

```
class TableItem : public QTableWidgetItem
{
public:
 TableItem();
 void setDigital(bool _is);

 void setDoubleText(const double _val, char *_mask);

private:
 bool digital;
};
```

3.2.2 Definición de métodos

Los nombres de los métodos son frases que incluyen verbos dado que los mismos generalmente son el producto de concatenar varias palabras, debiéndose emplear la primera palabra en minúscula, mayúscula para denotar la letra de inicio de cada una de las palabras restantes por las que esté formado y minúscula para las letras intermedias. Los atributos pasados por parámetro se separarán por una coma y un espacio después de ésta en caso de existir más de uno.A continuación se muestra un ejemplo de código de definición de método usado en la herramienta.

```
void MainWindow::treeClicked(QTreeWidgetItem * _item, int) {
 if ( _item->isDisabled() ) return;
 TreeItem * item = static_cast<TreeItem *>( _item );

 if ( item->getType() == FUNCTION ) ui->tabWidget->setCurrentIndex(1);
 else ui->tabWidget->setCurrentIndex(0);
 if ( ( item->getId() != 0 ) && ( item->getType() != FUNCTION ) ) {
 this->clearView();
 this->flagAdd = false;
 ui->add->setText("Actualizar");
 ui->add->setIcon( QIcon(":/resources/actions/edit") );
 this->updateViewPoint( static_cast<TreePoint *>( _item ) );
 } else
 if ( ( item->getId() != 0 ) && ( item->getType() == FUNCTION ) ) {
 this->clearView();
 this->flagAdd = false;
 ui->add_2->setText("Actualizar");
 ui->add_2->setIcon( QIcon(":/resources/actions/edit") );
 this->updateViewFunction( static_cast<TreeFunction *>( _item ) );
 } else {
 this->clearView();
 }
}
```

3.2.3 Estructuras de control

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

```
while ( iterfunctions != this->functions->end() )
 {
 TreeFunction * func = iterfunctions->second;

 if ( func->isDisabled() ) {
 iterfunctions++;
 continue;
 }
 FuncTask * task =
 static_cast<FuncTask *>( factory->getPeriodicTask( func->getTypeFunc(),
    func->getTime() ) );
 task->setId( func->getId() );
 double min = func->getMin().toDouble();
 double max = func->getMax().toDouble();
 task->setMinMax(min, max);
 if ( func->getTypeFunc() == DecDouble || func->getTypeFunc() ==
 DecInt ) task->setCurrent(max);
 else task->setCurrent(min);
 task->setStep( func->getStep() );
 this->functionsTask->insert( std::pair<QString,FuncTask *>(func->getName(),task) );

 iterfunctions++;
}
```

3.3 Diagramas de componentes

Los diagramas de componentes detallan los elementos físicos del sistema y sus relaciones. Los mismos son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación. Estos cubren la vista de implementación estática y se relacionan con los diagramas de clases ya que un componente suele tener una o más clases e interfaces. Los diagramas de componentes se pueden agrupar en paquetes así como los objetos en clases, además pueden existir entre ellos relaciones de dependencia. (Torres, 2006)

A continuación se muestra en la Figura 14 el diagrama de componente generado de la solución propuesta.

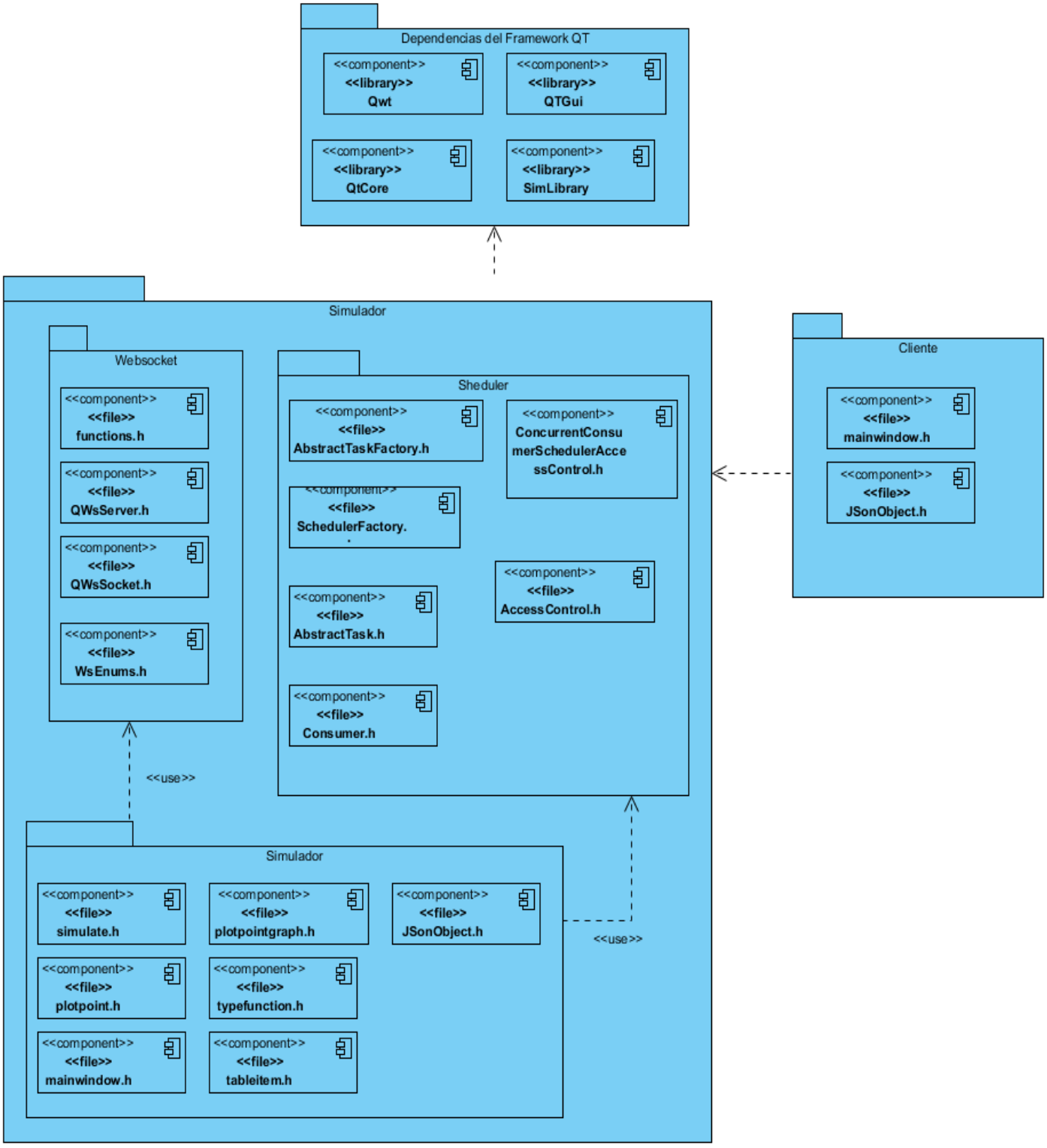

Figura 15: Diagrama de componentes

A continuación se describen los componentes del sistema utilizados como librerías.

- ***QtCore***: Biblioteca del framework Qt para la utilización de estructuras de datos (*QString, QList, etc*.)

- ***Qwt***: Biblioteca del framework Qt para la visualización de gráficas.
- ***QtGui***: Biblioteca del framework Qt que contiene todas las ventanas, botones y menús que se visualizan.
- ***SimLibrary***: Biblioteca creada para validar la simulación en una interfaz gráfica cliente la solución propuesta.

3.4 Diagrama de despliegue

El diagrama de despliegue es un modelo de objetos que describe la distribución física de un sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. En la figura 15 se realiza una representación gráfica de los nodos físicos en los que estará desplegado el sistema propuesto y la comunicación entre ellos. (Grady, 2000).

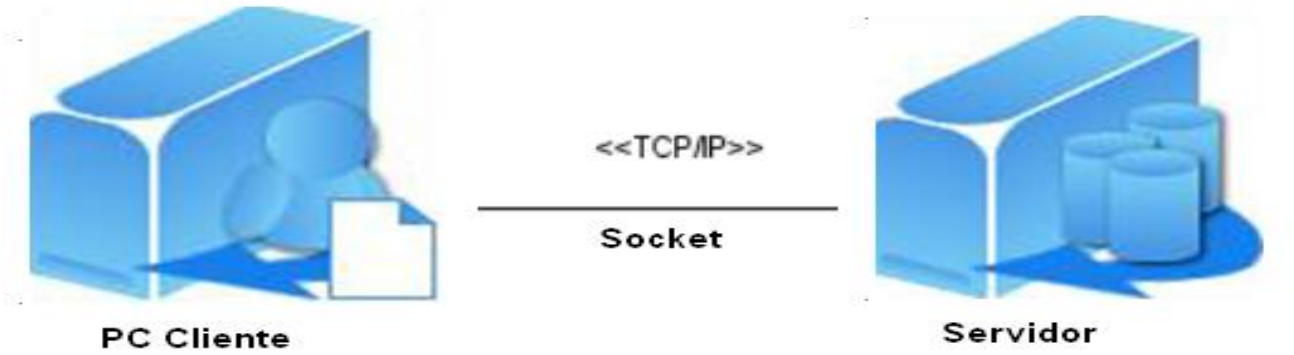

Figura 16: Diagrama de despliegue

Cliente:Es donde se va a desplegar la biblioteca SimLibrary de conjunto con el HMI.

Servidor:Es donde se va a desplegar la herramienta de simulación de datos.

3.5 Pruebas

Las pruebas de software son los procesos que permiten verificar y revelar la calidad de un producto software antes de su puesta en marcha. Básicamente, es una fase en el desarrollo de software que consiste en probar las aplicaciones construidas. Las pruebas se integran dentro de las diferentes fases del ciclo de vida del proyecto dentro de la Ingeniería de software. En este sentido, se ejecuta el aplicativo a probar y mediante técnicas experimentales se trata de descubrir qué errores tiene. Para determinar el nivel de calidad se deben efectuar unas medidas

o pruebas que permitan comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema. (Pressman, 2007).

3.5.1 Pruebas de aceptación

Las pruebas de aceptación no son más que pruebas funcionales sobre el sistema completo y buscan una cobertura de la especificación de requisitos y del manual del usuario. Estas pruebas no se realizan durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado (Pressman, 2007)

Una prueba de aceptación puede ir desde un informal caso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, las pruebas de aceptación pueden tener lugar a lo largo de semanas o meses, descubriendo así errores latentes o escondidos que pueden ir degradando el funcionamiento del sistema. Estas pruebas son muy importantes ya que definen el paso nuevas fases del proyecto como el despliegue y mantenimiento. (Pressman, 2007)

La ejecución de las pruebas previamente diseñadas permitió la evaluación de las funcionalidades de la aplicación desarrollada antes de implantarlo en su entorno real de explotación. Los resultados obtenidos se muestran a continuación:

Tabla 8: Caso de prueba Gestionar Variables

Caso de Prueba de Aceptación	
Número: 1	**Caso de Uso: Gestionar Variables**
Nombre: Crear variable analógica o digital.	
Descripción: Se comprueba la creación de las variables a través de la interfaz visual, tanto de tipo analógico como digital. En este caso se verifica que se creen la misma cantidad de variables que las especificadas por el usuario.	
Condiciones de Ejecución: Se debe seleccionar si la variable es analógica o digital.	

Pasos de Ejecución: Se agrega una variable con el nombre introducido por el usuario y se configuran sus propiedades. Se le agrega a esa variable una función de simulación previamente creada.
Resultado Esperado: Creación de la variable
Evaluación de la prueba: Satisfactoria

Tabla 9: Caso de prueba Ejecutar simulaciones.

Caso de Prueba de Aceptación	
Número: 2	**Caso de Uso: Ejecutar Simulaciones**
Nombre: Ejecutar Simulaciones	
Descripción: Proceso donde se visualizan las variables analógicas o digitales creadas con sus funciones y propiedades establecidas por el cliente.	
Condiciones de Ejecución: Existencia de variables analógicas o digitales creadas por el cliente.	
Pasos de Ejecución: Se crean las variables según su tipo y se configuran sus propiedades. Se le atribuye a dicha variable creada una función de simulación previamente creada. Se selecciona el botón de Iniciar Simulación de la barra de herramientas para comenzar el proceso de simulación. Seguidamente se observa que las variables habilitadas inician su simulación.	
Resultado Esperado: Simulación de las variables que se crearon anteriormente	
Evaluación de la prueba: Satisfactorio	

Tabla 10: Caso de prueba Salvar Configuraciones.

Caso de Prueba de Aceptación	
Número: 3	**Caso de Uso: Salvar Configuración**
Nombre: Salvar configuración en formato XML	
Descripción: Proceso donde se salvan las configuraciones en formato XML las variables analógicas o digitales creadas con sus funciónes y propiedades establecidas por el cliente en el sistema.	
Condiciones de Ejecución: Existencia de variables analógicas o digitales creadas por el cliente.	
Pasos de Ejecución: Se crean las variables según su tipo y se configuran sus propiedades. Se le atribuye a dichas variables creadas una función de simulación previamente creada. Se selecciona el botón Salvar SIM ubicado en la barra de herramientas. Seguidamente se observa que se abre una ventana donde el usuario guarda en la dirección deseada la configuración dándole el nombre que el mismo desee.	
Resultado Esperado: Salva exitosa.	
Evaluación de la prueba: Satisfactorio	

Tabla 11: Caso de prueba Cargar Configuraciones.

Caso de Prueba de Aceptación	
Número: 4	**Caso de Uso: Cargar Configuración**
Nombre: Cargar configuración en formato XML	
Descripción: Proceso donde se cargan las configuraciones en formato XML de las variables analógicas o digitales creadas con sus funciones y propiedades establecidas	

por el cliente en el sistema anteriormente guardadas.
Condiciones de Ejecución: Existencia de un archivo con variables analógicas o digitales creadas por el cliente en formato XML.
Pasos de Ejecución: Se selecciona el botón Cargar SIM ubicado en la barra de herramienta en la parte superior. Se abre una ventana para buscar el fichero anteriormente guardado. Se selecciona el fichero deseado y se visualiza todas las variables y funciones que se encontraban guardadas en el archivo.
Resultado Esperado: Visualización de todas las variables y funciones anteriormente guardadas
Evaluación de la prueba: Satisfactorio

3.5.2 Pruebas unitarias

Se aplican a un componente del software. Podemos considerar como Componente (elemento indivisible) a una función, una clase, una librería, etc. Estas pruebas las ejecuta el desarrollador cada vez que va probando fragmentos de código o scripts para ver si todo funciona como se desea. (Rodriguez, 2006)

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el fragmento de código debe satisfacer. Estas pruebas aisladas proporcionan cuatro ventajas básicas (Universidad Cibertec, 2011):

- **Fomentan el cambio**: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se llama refactorización) puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- **Simplifica la integración**: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente.

- **Documenta el código**: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
- **Separación de la interfaz y la implementación**: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.

En la siguiente tabla se muestra las pruebas unitarias realizadas a la solución propuesta:

Tabla 12: Prueba unitaria Graficar.

Prueba # 1	Graficar
Descripción	Verificar que el sistema es capaz de graficar el comportamiento de las variables analógicas y digitales.
Objetivo	Visualización del comportamiento de las variables en el rango determinado por sus propiedades
Condiciones	Existencia en el sistema de variables.
Resultado Esperado	Buen funcionamiento de la gráfica mostrando los datos de simulación en tiempo real.
Resultado Obtenido	La prueba fue satisfactoria pues los datos se mostraron con sus rangos definidos

Tabla 13: Prueba unitaria Conexión Servidor-Cliente .

Prueba # 2	Conexión Servidor-Cliente
Descripción	Verificar que el sistema es capaz de enviar la simulación de las variables a una interfaz cliente a través de una biblioteca.
Objetivo	Visualización del comportamiento de las

	variables que provienen del servidor hacia el cliente.
Condiciones	Existencia en el sistema de la biblioteca **SimLibrary**. Existencia de datos de simulación.
Resultado Esperado	Buen funcionamiento de la librería mostrando los datos de simulación en tiempo real en la interfaz cliente
Resultado Obtenido	La prueba fue satisfactoria pues la biblioteca logró conectar al servidor y al cliente para la transferencia de datos

3.5.3 Resultados de las pruebas

A continuación se muestra la **Tabla 7** el análisis de las no conformidades encontradas en cada iteración de las pruebas realizadas al sistema. Se comienza por la primera iteración encontrando 6 no conformidades de 14 requisitos funcionales relacionadas con errores de validación de datos. Luego se prosigue a una segunda iteración disminuyendo en esta a 4. Posteriormente se ejecuta la tercera y última iteración donde se eliminan todas las no conformidades, obteniendo un correcto funcionamiento de los componentes gráficos desarrollados.

Tabla 14: Análisis de las no conformidades

Iteración	No Conformidades
1ra	6
2da	4
3ra	0

Después de realizadas estas pruebas se puede concluir que la herramienta de simulación de datos cumple con los requerimientos funcionales y está lista para ser integrada en el Módulo Interfaz Hombre Máquina del SCADA GALBA.

3.6 Conclusiones Parciales

Durante el desarrollo de este capítulo se expusieron los principales artefactos del modelo de implementación lo que permitió mostrar los componentes del sistema y sus relaciones a través del diagrama de componentes. Mediante el diagrama de despliegue se modeló la arquitectura en tiempo de ejecución y se mostró los vínculos de comunicación entre los nodos. Además, se realizó el modelo de prueba para comprobar las funcionalidades identificadas durante el proceso de desarrollo del software y se describieron los casos de prueba que verifican el correcto funcionamiento del sistema a través de las pruebas unitarias y de aceptación.

Conclusiones Generales

Una vez finalizado el trabajo de diploma se llegó a las siguientes conclusiones:

- El análisis de las estructuras de los módulos del GALBA, especialmente el HMI, permitió el desarrollo de una herramienta para simular el comportamiento real de los procesos de supervisión y control, específicamente para las variables analógicas y digitales.
- El sistema desarrollado permite que varios ordenadores de un entorno de trabajo puedan utilizar una misma configuración compartida por uno de ellos.
- Las pruebas y validaciones realizadas permitieron evaluar la calidad y fiabilidad de la herramienta propuesta, demostrando la factibilidad de su uso en el desarrollo de las funcionalidades que serán ejecutadas posteriormente en el módulo de HMI.

Recomendaciones

- Vincular la biblioteca SimLibrary en el entorno de visualización del HMI SCADA Guardián del Alba y ejecutar el simulador para probar las funcionalidades del mismo.
- Implementarle a la herramienta nuevos tipos de variables para lograr simulaciones avanzadas.

Bibliografía

Alvarez, Miguel Angel. 2014. Desarrolladores Web. [Online] 2014. [Cited: Febrero 17, 2015.] http://www.desarrolloweb.com/articulos/que-es-mvc.html.

Automatas Industriales. 2000. [Online] 2000. http://www.automatas.org/redes/scada.htm.

Debian-es. 2010. Debian. [Online] 2010. [Cited: Diciembre 15, 2014.] http://es-debian.com.

Eclipse. 2014. IBM developerWorks. [Online] 2014. [Cited: Noviembre 25, 2014.] http://www.ibm.com/developerworks/ssa/library/os-ecov/.

Escolme Virtual. 2010. [Online] 2010. [Cited: Noviembre 25, 2014.] http://escolmevirtual.com/portal/index2.php?option=com_content&do_pdf=1&id=39.

Floyd, Michael. 2000. *Creación de sitios web con XML.* Madrid : Pearson Education, 2000.

GNU Operating System. 2015. [Online] 2015. http://www.gnu.org/home.es.html.

JACOBSON, GRADY RUMBAUGH,JAMES. 2000. *El Proceso Unificado de Desarrollo de Software.* s.l. : The Addison-Wesley Object Technology Series., 2000.

Jalon, Javier García de. 2009. *Aprenda C++ como si estuviera en primero.* s.l. : Escuela Superior de Ingenieros Industriales de San Sebastián.Universidad de Navarra, 2009.

Lopez. 2013. Visual Paradigm. [Online] 2013. http://www.ctr.unican.es/asignaturas/is1/is1-p01-trans01_s1&s2.pdf].

MADRID, U. P. D. 2012. *Patrones-del-Gang-of-Four.* 2012.

MARIN, M. D. A. Definición de lenguaje de programación, tipos y ejemplos. 2010. [Online] 2010. http://catedraprogramacion.foroactivos.net/t83-definicion-de-lenguaje-de-programacion-tipos-ejemplos.

Mis Respuestas. 2006. [Online] 2006. http://www.misrespuestas.com/que-es-una-metodologia.html.

Montero, Dagoberto. 2007. www.scribd.com. [Online] 2007. http://www.scribd.com/doc/13473499/Introduccion-a-Los-Sistemas-scada.

Pothamsetty, Venkat. 2011. [Online] 2011. [Cited: Diciembre 3, 2014.] http://scadahoneynet.sourceforge.net.

Pressman. Vol. 5ta Edicion.

—. 2007. *Ingeniería de Software un Enfoque Práctico.* 2007.

QT Creator. 2009. QT. [Online] 2009. [Cited: Octubre 30, 2014.] http://doc.qt.io/qtcreator/index.html.

QT Digia. 2012. [Online] 2012. [Cited: Enero 15, 2015.] http://qt.Digia.com/Product.

Qwt SouceForge. 2008. [Online] 2008. http://qwt.sourceforge.net.

2015. Sistema Operativo GNU. [Online] 2015. [Cited: Enero 13, 2015.] http://www.gnu.org/philosophy/free-sw.es.html.

Stroustrup. 2013. *.The C++ Programing Languaje. s.l.* s.l. : Addison-Wesley, 2013. ISBN 978-0321563842., 2013.

UAG. 2008. Universidad Autóctona de Guadalajara. [Online] 2008. [Cited: Noviembre 2, 2014.] http://genesis.uag.mx/edmedia/material/comuelectro/uni1_2_4.cfm.

UCLA. 2010. Universidad Centroccidental Lisandro Alvarado. [Online] 2010. [Cited: Octubre 3, 2014.] http://www.ucla.edu.ve/dac/Departamentos/coordinaciones/informaticai/documentos/PROCESAMIENTO%20DE%20DATOS.htm.

Universidad Cibertec. 2011. *Pruebas de Software.* 2011.

Universidad de Chile. 2010. Facultad de Ciencias Fisicas y Matematicas. [Online] 2010. http://www.scribd.com/doc/37508751/Manual-Modsim.

Valle, Jose Guillermo. 2009. Monografias. [Online] 2009. [Cited: Marzo 23, 2015.] http://www.monografias.com/trabajos24/arquitectura-cliente-servidor/arquitectura-cliente-servidor.shtml.

TORRES, Patricio Letelier. Desarrollo de Software Orientado a Objeto usando UML. Universidad Politecnica de Valencia (UPV)–España, **2004.**

BOOCH, Grady, El lenguaje unificado de modelado. Addison-Wesley, 2005.

RODRIGUEZ, **Jorge**. Pruebas unitarias. 2006.