Facultad 4

Centro de Informática Industrial

Tema: Componentes gráficos para la representación de signos de advertencia y señales de dirección en el SCADA SAINUX 2.0.

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

Autor: Yosvani Sánchez Ojeda

Tutor: Ing. Henry Marcelo Cabrera

Co-Tutor: Ing. Adolfo Yasser Santana

Co-Tutora: Ing. Claudia González Fernández.

Curso docente: 2016-2017.

Declaración de Autoría

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

Para que así conste se firma la presente a los _____ días del mes de _____ del año _____.

Firma del Autor: Yosvani Sánchez Ojeda

Firma del Tutor: Henry M. Cabrera Robles

Firma del Co-Tutor: Adolfo Yasser Santana

Firma del Co-tutor: Claudia González Fernández


El genio se hace con
un 1% de talento,
y un 99% de trabajo.
Albert Einstein

DATOS DEL CONTACTO

Tutor: Ing. Henry Marcelo Cabrera Robles.
Edad: 26
Ciudadanía: cubana
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Ciencias Informáticas
Síntesis del Tutor: Graduado de Ingeniería en Ciencias Informáticas en julio del 2013, en la Universidad de Ciencias Informáticas. Posee 4 años de experiencia en el centro de informática industrial CEDIN, el cual ejerce el cargo de jefe de proyecto del SCADA SAINUX2.0.
E-mail: hmcabrera@uci.cu

Co-Tutor: Ing. Adolfo Yasser Santana Rojas.
Edad: 25
Ciudadanía: cubana
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Ciencias Informáticas
Síntesis del Tutor: Graduado de Ingeniería en Ciencias Informáticas en julio del 2015, en la Universidad de Ciencias Informáticas. Posee 2 años de experiencia en el centro de informática industrial CEDIN. En el cual es trabajador del mismo.
E-mail: aysantana@uci.cu

Co-Tutora: Ing. Claudia María González Fernández.
Edad: 24
Ciudadanía: cubana
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Ciencias Informáticas
Síntesis del Tutor: Graduado de Ingeniería en Ciencias Informáticas en julio del 2016, en la Universidad de Ciencias Informáticas. Posee 1 año de experiencia en el centro de informática industrial CEDIN, en el cual es ejerce como trabajadora del mismo.
E-mail: cmfernandez@uci.cu

AGRADECIMIENTOS

Muchas personas fueron participes de la realización de este trabajo, a todas ellas les ofrezco mis más sinceros agradecimientos en especial a:

Mi mamá y mi papá que además de darme la vida, supieron hacer de mí una persona de bien, me enseñaron a luchar por las cosas que verdaderamente quiero, a ser fuerte ante las situaciones de la vida, a no rendirme nunca a pesar de las circunstancias, por confiar en mí en cada momento, por darme esa confianza para contarle cada problema que tuviera y siempre apoyarme, le agradezco a la vida por darme los mejores padre, que los quiero, gracias por existir.

A mi Abuela Caridad que dicen en mi casa que soy su niño lindo, que la quiero mucho y que si pudiera que viviera 80 años más, gracias por apoyarme siempre.

A mi novia que en cada momento desde que la conocí siempre ha estado a mi lado apoyándome, ayudándome en cualquiera de las circunstancias que la quiero mucho y le doy gracias a la vida por haberla conocido, también a sus padres que son personas maravillosas y me han apoyado siempre.

A mi hermana linda que ella sabe que la quiero mucho y es muy importante en mi vida y al resto de mi familia, a mis tíos y abuelos, primos que somos una familia maravillosa.

A mis tutores por la gran ayuda que me brindaron durante el desarrollo de mi tesis, a Henry porque más que un tutor lo considero amigo, siempre estuvo en los malos y buenos momentos dándome consejos ayudándome cada vez que tuviera alguna duda, a Claudia por guiarme en el transcurso de la

realización de la tesis y apoyarme en cada momento.

A todos mi amigos y compañeros del aula, a todos mis compañeros del duty, del futbol, la pelota y del edificio, en especial a Miguel, Oscar, Jorgito y Richard, Jose, Rogelio, Yasiel, Oscar el pelú, por brindarme su ayuda y por compartir conmigo todos los buenos y malos momentos durante 5 años a lo largo de esta carrera.

A *mi facultad, que me ha dado ese orgullo de decir "soy de la 5", en donde he vivido un ambiente de armonía y he conocido a verdaderos amigos aunque esta facultad ya no exista siempre estará en mi corazón.*

A todos los que de una forma u otra lograron que mi sueño de ser profesional se convirtiera en realidad, gracias.

DEDICATORIA

Quiero dedicar este trabajo a mi abuela Caridad, a mi papá y mamá que han sido el motor impulsor de mi vida, la persona que jamás me ha dado la espalda, siempre ha confiado en mí y por creer de que si podía ser ingeniero informático y superarme profesionalmente.

RESUMEN

En los últimos años la automatización de procesos industriales se ha convertido en un gran avance para el sector empresarial. Dentro de las aplicaciones informáticas que se utilizan con este fin, se encuentran los sistemas de supervisión, control y adquisición de datos, conocidos como SCADA.

Diversas industrias emplean para la automatización de sus procesos productivos los sistemas SCADA, acrónimo de *Supervisory Control and Data Acquisition* (Control, Supervisión y Adquisición de Datos).

El objetivo de este trabajo, es el desarrollo de componentes gráficos reutilizables, con nuevas funcionalidades, que puedan ser configurados en cualquier editor gráfico que soporte Qt, y que puedan ser modificadas sus propiedades en ambiente de configuración. Para cumplir este objetivo se hizo un estudio de los conceptos, herramientas y tecnologías existente acerca del desarrollo de componentes gráficos para la visualización de procesos industriales.

Palabras claves: HMI, SCADA

ÍNDICE

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Con el continuo progreso de los avances tecnológicos que ha tenido la humanidad, los procesos industriales son cada vez más complejos y difíciles de manejar, llevan consigo una gran cantidad de operaciones que deben ser controladas a tiempo completo. Con el desarrollo de la informática, se ha hecho posible desarrollar software capaz de automatizar estos procesos, marcando una notable diferencia en eficiencia y resultados de las industrias. La informática industrial es la rama de esta ciencia encargada del tratamiento automático de la información proveniente de los procesos industriales, utilizando para ello computadoras. Los datos son introducidos a un ordenador, adquiriendo significado cuando se hace una interpretación de los mismos.

La Universidad de las Ciencias Informáticas (UCI), desarrolla diversos proyectos productivos para la automatización de instituciones dentro y fuera del país. Con la misión de desarrollar productos y servicios informáticos de automatización industrial con un alto valor agregado y que cumplan las necesidades y expectativas de los clientes, potenciando la formación especializada y la investigación. Tal es el caso del proyecto Continuidad del Sistema de automatización Industrial UX (SAINUX 2.0), desarrollado por el Centro de Informática Industrial (CEDIN) perteneciente a la Facultad 4.

Un sistema de supervisión, control y adquisición de datos SCADA monitorea y controla un sitio completo ya que se puede extender sobre una gran distancia (kilómetros / millas). La instalación de un sistema SCADA necesita un hardware de señal de entrada y salida, sensores y actuadores, controladores, HMI, redes, comunicaciones y base de datos.

Los módulos que componen un SCADA son Configuración, Interfaz gráfica del operador, Módulo de proceso, Gestión y archivo de datos y Comunicaciones. El módulo Interfaz Hombre Maquina (HMI por sus siglas en ingles), en el SCADA se encarga de representar, en un ordenador, los procesos que ocurren en el campo, muestra los componentes implicados, los sensores, las estaciones remotas, y el sistema de comunicación dándole al operador total control. Este módulo es el que permite al operador estar en contacto directo con el sistema, realizar la supervisión y el control del proceso en general.

El HMI trabaja sobre dos entornos: el entorno de configuración (EC), donde los mantenedores configuran la información específica del área que se desea supervisar y diseñan los despliegues, los cuales haciendo uso de los componentes gráficos permiten simular los procesos de campo y el entorno de visualización (EV), donde el operador puede

supervisar y controlar la configuración realizada en el EC, interactuando con los componentes gráficos para emitir control sobre el sistema, monitorear los cambios de estado de las variables, gestionar alarmas, generar reportes.

Tanto para el EC como para el EV, el HMI provee una biblioteca de componentes gráficos (CG) que permiten recrear de una manera lo más real posible los procesos de campo. Estos gráficos pueden ser simples figuras geométricas como: línea, rectángulo, elipse, texto, polígono, poli línea, curvas; pero también muy complejas como los componentes de hardware utilizados en la industria que se desean supervisar.

Entre los componentes usados en un proceso es necesario la representación de Signos de advertencia y Señales de dirección. La biblioteca de componentes gráficos del HMI SAINUX no le provee al mantenedor componentes para dicha representación. Para solventar dicha situación el mantenedor utiliza o combina varias figuras geométricas simples como poli línea, curvas, lo cual tiene como desventajas:

- La imagen en la mayoría de los casos no es la representación fiel de la red que se desea supervisar.
- La imagen sufre pérdida de la calidad, cuando se redimensiona, afectando negativamente la representación gráfica.
- No permite la personalización de la red al mantenedor.
- La representación con figuras simples hace engorroso el proceso de configuración del proceso a supervisar, puede introducir errores y el sistema pierde usabilidad.

Para dar solución a esta problemática se propone como **problema de la investigación:** ¿Cómo contribuir a elevar los grados de representación de los Signos de advertencia y Señales de dirección en la Interfaz Hombre Máquina del sistema SCADA SAINUX?

El objeto de estudio se centra en: Proceso de visualización y configuración de componentes gráficos para procesos industriales.

Delimitando como **campo de acción**: Componentes gráficos para la representación de signos de advertencia y señales de dirección en la Interfaz Hombre Máquina del sistema SCADA SAINUX 2.0.

Teniéndose como **Objetivo general:** Desarrollar componentes gráficos que permitan elevar los grados de representación de signos de advertencia y señales de dirección en el proceso

de visualización de la Interfaz Hombre Máquina del sistema SCADA SAINUX 2.0.

Posibles resultados:

1. Componentes gráficos para la representación de signos de advertencia.
2. Componentes gráficos para la representación de flechas de dirección.
3. Paleta de componentes gráficos especializada en signos de advertencia.
4. Paleta de componentes gráficos especializada en flechas de dirección.

Tareas a cumplir por el estudiante:

1) Elaboración del marco teórico de la investigación a través del estudio del estado del arte que existe actualmente sobre el tema.
2) Identificación de los principales elementos que representan signos de advertencia y señales de dirección en los procesos industriales y su representación en sistemas SCADA.
3) Caracterización de los principales elementos que representan signos de advertencia y señales de dirección en los procesos industriales y su representación en sistemas SCADA.
4) Realización del levantamiento de requisitos funcionales y no funcionales.
5) Implementación de componentes gráficos que brinden solución al problema planteado.
6) Realización de pruebas para validar el cumplimiento de los requerimientos.

En el desarrollo del presente trabajo se utilizan los siguientes métodos de investigación:

Métodos Teóricos

- **Analítico-sintético:** Para el estudio de los conceptos empleados en los sistemas SCADA, analizando todos los documentos elaborados, para la extracción de los elementos más importantes.
- **Histórico-lógico:** Para la comprensión de los antecedentes y las tendencias actuales referidas la evolución en el mundo de los sistemas SCADA web.
- **Modelación:** Para la elaboración de múltiples diagramas para un mejor entendimiento del problema y solución.

Métodos Empíricos

- **Observación:** Se puso en práctica este método, para conocer el funcionamiento existente en los despliegues del proyecto Continuidad del Sistema de automatización Industrial UX (SAINUX 2.0), mediante el comportamiento de los dispositivos de campo en las propiedades de los objetos gráficos y sumarios empleados para la toma de decisiones de los operadores.
- **Consulta bibliográfica:** Empleada para consultar las fuentes de información relacionadas con los tipos de los sumarios y objetos gráficos visualizados en los entornos de escritorio de los HMI en sistemas SCADAS.

El presente documento está estructurado en tres capítulos:

Capítulo 1. Fundamentación teórica: Se realiza un esbozo teórico centrado en los conceptos y características fundamentales de los componentes gráficos para el proyecto Continuidad del Sistema de automatización Industrial UX (SAINUX 2.0) para la representación de símbolos de advertencias y señales de dirección.

Capítulo 2. Construcción de la solución: Se analiza la solución propuesta, se realiza el levantamiento de requisitos funcionales y no funcionales apoyándonos en la metodología de desarrollo escogida.

Capítulo 3. Implementación y prueba: En este capítulo se describe la fase de implementación del sistema, según la metodología propuesta. Se realiza la fase de pruebas, a la cual se somete el producto para validar su correcto funcionamiento, permitiendo comprobar que el mismo cumple con todos los requerimientos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se hace un estudio del estado del arte, partiendo de los conceptos asociados a un sistema SCADA. Se explican los módulos y el funcionamiento de un sistema SCADA para la gestión y control de datos, enfatizando en el módulo HMI. Se describen también las tecnologías que se utilizaron para construir el software, lenguaje de programación y herramientas de desarrollo para darle solución al problema planteado.

1.2 Sistemas SCADAS

El SCADA es un sistema de tiempo real, distribuido en módulos que trabajan de manera conjunta posibilitando el funcionamiento del sistema como un todo. Estos módulos se encuentran interconectados a través de un software para la distribución de los servicios en la red, conocido como software de comunicación entre aplicaciones.(1)

1.2.1 Generalidades de un Sistema SCADA

Las características principales de un SCADA son las siguientes:

- Adquisición y almacenado de datos para recoger, procesar y almacenar la información recibida en forma continua y confiable.(1)
- Representación gráfica y animada de variables de proceso y su monitorización por medio de alarmas.(1)
- Ejecutar acciones de control para modificar la evolución del proceso, actuando ya sea sobre los reguladores autónomos básicos (consignas, alarmas, menús) o directamente sobre el proceso mediante las salidas conectadas.(1)
- Arquitectura abierta y flexible con capacidad de ampliación y adaptación.
- Supervisión, para observar desde un monitor la evolución de las variables de control.(1)
- Transmisión de información con dispositivos de campo y otros PC.
- Base de datos, gestión de datos con bajos tiempos de acceso.
- Presentación, representación gráfica de los datos. Interfaz del Operador o HMI.(1)

- Explotación de los datos adquiridos para gestión de la calidad, control estadístico, gestión de la producción y gestión administrativa y financiera.
- Alertar al operador sobre cambios detectados en la planta, tanto aquellos que no se consideren normales (alarmas) como los que se produzcan en su operación diaria (eventos). Estos cambios son almacenados en el sistema para su posterior análisis.(1)

1.2.2 Funciones de un Sistema SCADA

- **Monitoreo:** Es la habilidad de obtener y mostrar datos en tiempo real. Estos datos se pueden mostrar como números, texto o gráficos que permitan una lectura más fácil de interpretar.(2)
- **Supervisión:** Esta función permite junto con el monitoreo la posibilidad de ajustar las condiciones de trabajo del proceso directamente desde la computadora. Alarmas: Es la capacidad de reconocer eventos excepcionales dentro del proceso y reportar estos eventos. Las alarmas son reportadas basadas en límites de control preestablecidos.(2)
- **Control:** Es la capacidad de aplicar algoritmos que ajustan los valores del proceso y así mantener estos valores dentro de ciertos límites. Este va más allá del control de supervisión, removiendo la necesidad de la interacción humana. Sin embargo, la aplicación de esta función desde un software corriendo en una PC puede quedar limitada por la confiabilidad que quiera obtenerse del sistema.(2)
- **Históricos:** Es la capacidad de muestrear y almacenar en archivos, datos del proceso a una determinada frecuencia. Este almacenamiento de datos es una poderosa herramienta para la optimización y corrección de procesos(2).

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

- **Configuración:** permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar.(1)
- **Interfaz Hombre Máquina:** proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado en el SCADA o importados desde otra aplicación durante la configuración del paquete.(2)

- **Módulo de proceso:** ejecuta las acciones de mando pre programado a partir de los valores actuales de variables leídas.
- **Gestión y archivo de datos:** se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos(2)
- **Comunicaciones:** se encarga de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA, y entre esta y el resto de elementos informáticos de gestión.(2)

1.2.3 Módulo Interfaz Hombre Máquina

La Interfaz Hombre-Máquina o HMI (Human Machine Interface) es el sistema que presenta los datos a un operador (humano) y a través de la cual este controla el proceso. Son interfaces gráficas que muestran información del proceso en tiempo real, utilizando diagramas esquemáticos, algunos contornos y hasta animaciones en pantalla o paneles.(2)

Los sistemas HMI facilitan mucho las tareas de diseño debido a que incorporan protocolos para comunicarse con los dispositivos de campo más conocidos, tienen herramientas para crear bases de datos dinámicas, permiten crear y animar pantallas de forma sencilla y además incluyen gran cantidad de librerías de objetos para representar los diferentes dispositivos de la industria como motores, tanques, indicadores, interruptores válvulas y bomba, Signos de advertencias, Señales de dirección, cámara de seguridad, entre otras.(3)

1.2.3.1 Entornos de Trabajo para el Módulo HMI

En el SCADA SAINUX, el módulo HMI presenta dos ambientes, el ambiente de edición que permite configurar los procesos, definir y gestionar las variables, editar los controladores, los comandos, las alarmas y opciones adicionales. En este entorno las pantallas o procesos definen sus relaciones entre ellas, determinando el orden de aparición, los drivers, los comandos, las alarmas y las variables a visualizar que después se procesarán y controlarán en forma de listas o tablas para una mejor gestión.(4)

Por otra parte, el editor es el que permite a un operador definir el ambiente de trabajo del SCADA, adaptándolo mejor a la aplicación particular que se desea desarrollar. Este entorno visualiza los datos históricos en forma de gráficos de tendencia y hace manejo de alarmas y eventos. Es el encargado de supervisar de manera directa el proceso puesto que interactúa con los operadores.(4)

Algunas de las funcionalidades con las que un HMI debe contar son:

- Visualizar las variables mediante pantallas con objetos animados.
- Permitir que el operador pueda enviar señales al proceso mediante botones, controles on/off o ajustes continuos con el mouse o el teclado.
- Supervisar niveles de alarma y alertar/actuar en caso de que las variables excedan los límites normales.
- Almacenar los valores de las variables para un análisis estadístico y/o control.
- Controlar en forma limitada ciertas variables de proceso.

1.2.3.2 Componentes Gráficos

Los componentes gráficos son un conjunto de elementos que presentan como objetivo la comunicación visual a partir de aplicaciones que tienen su origen en el diseño bidimensional. Un buen empleo de los gráficos determina el comportamiento y el seguimiento del objeto en profundidad.

En esencia los gráficos representan un lenguaje visual que te permite explorar, dar sentido y comunicar información cuantitativa, se pueden incorporar a todo tipo de soporte y material facilitando la comunicación global y la comprensión.

1.3 Componentes gráficos signos de advertencia y señales de dirección.

En un sistema SCADA es muy importante la representación de Signos de advertencia y señales de dirección; ya que son componentes gráficos que permiten recrear desde el entorno de configuración y del entorno de visualización; el conjunto de estímulos que condicionan la actuación del individuo que los recibe frente a circunstancias de riesgos y protecciones necesarias a utilizar, que pretenden resaltar algún cambio o problema que haya ocurrido en el sistema, emitiendo señales a través de imágenes de advertencias y especificando mediante las flechas de dirección. Se utilizan generalmente en empresas con sistemas que emplean en el ambiente de trabajo software para detectar disímiles variedades de peligros como:(5)

- Riesgo de incendio (materiales inflamables).
- Riesgo de explosión (materias explosivas).
- Riesgo de radiación (material radiactivo).
- Riesgo de intoxicación (sustancias toxicas).

- Riesgo de electricidad (electricidad).
- Riesgo de explosión.
- Riesgo de láser.
- Riesgo de combustible.
- Riesgo de radiaciones no ionizante.
- Riesgo de peligro general.
- Riesgo de material nocivo o irritante.

Un sistema de información de seguridad que se basa en el uso de componentes, se rigen por las normas internacionales desarrolladas por ISO (Organización Internacional de Normalización) proporcionando a las personas de todo el mundo, un conjunto coherente de símbolos gráficos para ayudar a superar barreras lingüísticas; evitando con ello la proliferación de símbolos gráficos que puedan producir confusión a los consumidores. Proporcionando información de forma compacta, además de guiar al público a un resultado deseado o una decisión apropiada. Para el desarrollo de estos componentes solo se tuvo en cuenta el estudio de las normas (ISO 3864-1) y estándares existentes para desarrollar componentes gráficos lo más comunes posibles, que tuvieran las características de los ya existentes. Para ello se tomaron los siguientes términos:(5)

- **Color de seguridad:** Un color al que se atribuye una significación determinada en relación con la seguridad y salud en el trabajo. Fondo amarillo. Franja triangular negra. El símbolo de seguridad será negro y estará colocado en el centro de la señal, la franja periférica amarilla es opcional. El color amarillo debe cubrir por lo menos el 50% del área de la señal.
- **Señal de advertencia o peligro:** Señal que advierte de un riesgo o peligro. Advierten sobre los peligros que podrían resultar en lesiones personales o amenazas para la salud.

1.3.1 Signos de Advertencia

En la industria se emplea un sistema especial de diferentes tipos de símbolos con el objetivo de trasmitir de una forma más fácil y especifica la información. Esto es indispensable en el diseño, selección, operación y mantenimiento de los sistemas de control.

Los símbolos de advertencia, también llamados Señalamientos de Seguridad, desempeñan un papel vital en la comunicación de la información en materia de seguridad. Pueden reducir al mínimo el riesgo de un accidente que ocurre en un lugar de trabajo y son una manera fácil y comprensible universalmente de conseguir hacer llegar su mensaje a todo el mundo. No obstante, los empleadores deben proporcionar información a los empleados sobre el significado y los requisitos de todos los signos utilizados en el lugar de trabajo.(6)

La correcta señalización resulta eficaz como técnica de seguridad, pero no debe olvidarse que por sí misma nunca elimina el riesgo. Las señales deberán retirarse cuando deje de existir la situación que las justificaba.(6)

1.3.2 Señales de Dirección

Una Señal es por lo general, una representación gráfica que indica o tiene un propósito como guía en cualquier camino o función. Una señal corresponde a las funciones de un indicador, puesto que una señal refleja cual es la acción que se debe realizar.

La presencia de señales en cualquier acción implica la puesta en escena de íconos, imágenes y ejercicios para informar cual es la situación que se vive. Las señales como aplicación, son una herramienta netamente sensorial, ya que se perciben con los sentidos y son analizadas y asociadas. En ocasiones, las señales no corresponden directamente con la acción que se debe ejecutar, pero la convención y la costumbre permiten una correcta interpretación de estas.(7)

1.4 Herramientas y Tecnologías

Las herramientas y tecnologías escogidas para el desarrollo de un proyecto deben ser seleccionadas cautelosamente, ya que pueden suponer el fracaso de éste o pueden aumentar su complejidad, para lo cual se deben conocer cuáles son las distintas alternativas y las necesidades del proyecto. Se hace una breve descripción del ambiente de trabajo como es el sistema operativo y las herramientas y tecnologías seleccionadas. La selección de varias herramientas y tecnologías está fundamentada por su utilización en el desarrollo del proyecto Continuidad del Sistema de Automatización Industrial UX (SAINUX 2.0).

1.4.1 Gráficos Vectoriales Estables (SVG)

SVG son las siglas de *Scalable Vector Graphics*, que podríamos traducir libremente al

español como gráficos basados en vectores escalables. En otras palabras, decir que es un formato gráfico basado en XML para crear archivos vectoriales en 2D, con un lenguaje de marcado por medio de etiquetas. Entre sus posibilidades, podemos señalar la capacidad de usar tres tipos de objetos gráficos: formas de vectores gráficos (entre las que se incluyen líneas, polígonos, poli línea, rectángulos, círculos o elipses), imágenes y texto. Además, a los objetos gráficos les podemos aplicar transformaciones (traslaciones, escala), animaciones y efectos de filtro (muy variados, como los que conocemos en programas de diseño como *Photoshop*). Algunas ventajas de este formato son: no pierde calidad si hacemos zoom, se puede escalar, se muestra de forma progresiva, no se tiene que esperar a que se descargue todo el archivo, los vectoriales por ser solamente parámetros matemáticos, suelen ser mucho menos pesados que una imagen rasterizada o de píxeles. Los gráficos vectoriales pueden ser transformados (estirar, rotar, mover, distorsionar) de una manera más sencilla y con menos requerimientos de memoria en el equipo.(8)

1.4.2 Inskcape

Una herramienta sencilla y potente para poder realizar dibujos, ilustraciones o diseños es *Inkscape*. Este es un “editor de gráficos vectoriales de código abierto, con capacidades similares a Ilustrato*r,* Freehand, CorelDraw o Xara X, usando el estándar de la W3C: el formato de archivo Scalable Vector Graphics (SVG)”, soportando formas, trazos, texto, marcadores, clones, mezclas de canales alfa, transformaciones, gradientes, patrones y agrupamientos. Con este programa podemos tener una herramienta de dibujo potente y cómoda, compatible con los estándares XML, SVG y CSS. Uno de sus potenciales es, además de ser software libre, ser multiplataforma, es decir, que es compatible con diferentes sistemas operativos como GNU/Linux, Mac OS X y Windows. Estas características le proporcionan entre otras ventajas contar con una amplia red de usuarios y comunidades en las que puedes participar o solicitar ayuda para la realización de tus creaciones digitales.(9)

1.4.3 Sistema Operativo

Debian GNU/Linux es un sistema operativo de libre distribución y uso. Permite a varios usuarios acceder al mismo tiempo a través de terminales, y distribuye los recursos disponibles entre todos. En los sistemas operativos libres el software se distribuye en forma de paquetes modulares que suelen incluir completa información sobre interdependencias lo que facilita la gestión de la instalación del software, su actualización y su integridad. Con

frecuencia estos sistemas son mucho más avanzados que las alternativas ofrecidas por el software propietario. Este ha sido desarrollado por miles de usuarios de computadores a través del mundo y su licencia tiene por objeto asegurar que Linux siga siendo gratuito y a la vez estándar. Puede correr en la mayoría de plataformas del mercado (procesadores de la gama Intel y AMD, Motorola, Sun, Sparc). La versión empleada en la solución es 7.0 Wheezy.(10)

1.4.4 Lenguaje de Programación

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina. A pesar de que es un lenguaje orientado a objetos, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para la programación a bajo nivel. Esto nos permite utilizarlo para escribir controladores de dispositivos y otro software que se basen en la manipulación directa de hardware, bajo restricciones de tiempo real. C++ fue diseñado para que cada característica del lenguaje se pudiera utilizar en virtud de limitaciones severas de tiempo y espacio.(11)

1.4.5 Marco de Trabajo

Un *framework* es una abstracción de código común que provee funcionalidades genéricas que pueden ser utilizadas para desarrollar aplicaciones de manera rápida, fácil, modular y sencilla, ahorrando tiempo y esfuerzo. Qt es un *framework* multiplataforma orientado a objetos, se utiliza para desarrollar software que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario. Incluye clases, bibliotecas y herramientas para la producción de aplicaciones usando el lenguaje C++ de forma nativa. Es multiplataforma ya que funciona en las principales plataformas y tiene un amplio apoyo operando en varios sistemas como Unix (Linux, MacOS X, Solaris) o incluso toda la familia de Windows. El principal motivo por el que Qt se ha hecho tan popular es por su métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros métodos para el manejo de ficheros, además de estructuras de datos tradicionales.(12)

1.4.6 Entorno Integrado de Desarrollo

Para el desarrollo de este producto se requiere de un entorno de desarrollo integrado (IDE, por sus siglas en inglés), el cual puede denominarse como un entorno de programación que consiste en un editor de código y un compilador.

El IDE seleccionado para el desarrollo de la aplicación es Qt Creator en su versión 2.5.0. Qt Creator es un IDE multiplataforma para el desarrollo de aplicaciones que pueden o no tener interfaz gráfica. Este se centra en proporcionar características que ayudan a los nuevos usuarios del IDE a aprender y comenzar a desarrollar rápidamente.(13)

Existen otras características importantes que presenta este IDE como la disponibilidad de código fuente, la excelente documentación organizada que provee en el *Qt Assistant* y un editor para el diseño de formularios denominado *Qt Designer*.(13)

Qt Creator cuenta con:

- Un editor de código con soporte para C++.
- Herramientas para la rápida navegación por el código.
- Resaltado de sintaxis y auto-completado de código.
- Control estático de código y estilo a medida.
- Soporte para refactorización de código.
- Paréntesis coincidentes y modos de selección.

1.4.7 Herramienta Case

Las herramientas CASE (*Computer Aided Software Engineering*), en español Ingeniería de Software Asistida por Computadora, son aplicaciones informáticas utilizadas en el proceso de desarrollo de software en tareas como: realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente a partir del diseño, compilación automática, documentación o detección de errores entre otras.
Visual Paradigm es una herramienta CASE profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Este software ayuda a una más rápida construcción de aplicaciones de calidad y a un menor coste.

Las características principales de esta herramienta son las que a continuación se enuncian:(14)

- Soporte de UML.
- Ingeniería inversa – Código a modelo, código a diagrama.
- Generación de código – Modelo a código, diagrama a código.
- Generación de bases de datos -Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos -Desde Sistemas Gestores de Bases de Datos existentes a diagramas de Entidad-Relación.
- Editor de figuras.

1.5 Metodología de Desarrollo

Actualmente los negocios operan en un entorno global que cambia rápidamente. Tienen que responder a nuevas oportunidades y mercados, condiciones económicas cambiantes y la aparición de productos y servicios competidores. El software es parte de casi todas las operaciones de negocio, por lo que es fundamental que el software se desarrolle rápidamente para aprovechar nuevas oportunidades y responder a la presión competitiva. Actualmente el desarrollo y entrega de manera rápida son los requerimientos más críticos de los sistemas.(15)

Desarrollar un buen software depende de un sinnúmero de actividades y etapas, donde se debe elegir la mejor metodología para un equipo en un determinado proyecto, lo cual es trascendental para el éxito del producto.

Al no existir una metodología de software universal toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, tiempo, recursos) exigiéndose así que el proceso sea configurable. Estas se pueden dividir en dos grupos de acuerdo con sus características y los objetivos que persiguen: ágiles y robustas.(16)

Las metodologías ágiles tienen como principal característica, el desarrollo rápido de software, permiten al equipo adaptar y coordinar tareas, por la necesitad de satisfacer al cliente con la mayor brevedad posible. Deben cumplirse una serie de requerimientos dentro del equipo de desarrollo para poder aplicar un modelo como este.(17)

Las metodologías tradicionales en cambio, se centran especialmente en el control del proceso, mediante una exhaustiva documentación; definiendo roles, actividades, artefactos,

herramientas y notaciones para el modelado y una documentación detallada. Estas metodologías son muy efectivas y necesarias en proyectos grandes.(18)

Se presenta una tabla que ilustra las diferencias entre las metodologías tradicionales y ágiles.

Tabla 1"Diferencias entre metodologías Tradicionales y Ágiles".(19)

Metodologías Tradicionales	Metodologías Ágiles
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.	Basadas en heurísticas provenientes de prácticas de producción de código.
Resistencia a los cambios.	Preparadas para cambios durante el proyecto.
Impuestas externamente.	Impuestas internamente (por el equipo).
Proceso más controlado, con numerosas políticas/normas.	Proceso menos controlado, con pocos principios.
Existe un contrato prefijado.	No existe contrato tradicional (bastante Flexible).
El cliente se comunica con el equipo de desarrollo mediante reuniones.	El cliente es parte del equipo de desarrollo.
Grupos grandes y distribuidos.	Grupos pequeños (menos de 10 integrantes) trabajando en el mismo sitio.
Más artefactos.	Pocos artefactos.
Más roles.	Pocos roles.
La arquitectura de software es esencial y se expresa mediante modelos.	Menos énfasis en la arquitectura del software.

Por los recursos, la documentación necesaria, el escaso tiempo para la realización del

proyecto, y personal, se decide escoger una metodología ágil. Dentro de las metodologías ágiles existentes encontramos algunas como:

- XP.
- OPEN UP.
- BPM.
- DAC.
- KIMBALL.
- SXP.
- SCRUM.
- NOVA OPEN UP.
- AUP

Existen otras metodologías llamadas híbridas, tal es el caso de la AUP (*Agile Unified Process*, Proceso Unificado Ágil). Las metodologías llamadas híbridas caen dentro de alguna de las dos clasificaciones mencionadas: ágiles o tradicionales.

AUP es una versión simplificada de RUP (*Rational Unified Process*), que describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

AUP aplica técnicas ágiles incluyendo:

- Desarrollo dirigido por pruebas.
- Modelado ágil.
- Gestión de Cambios ágil.
- Refactorización de Base de Datos para mejorar la productividad.

Metodología AUP-UCI.

Para eliminar los errores encontrados se decidió converger a una metodología que cubra las particularidades de cada una de las ya aplicadas. Esta metodología escogida se adapta a lo que ya la Universidad ha estado proponiendo como ciclo de vida de los proyectos, sin alejarse de lo que hasta el momento se ha trabajado e introducir la menor cantidad de cambios posibles. Esta metodología propuesta es una variación al Proceso Unificado Ágil.(20)

El Proceso Unificado Ágil (AUP, por sus siglas en inglés) es una versión simplificada del Proceso Racional Unificado (RUP, por sus iniciales en inglés). 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.(20)

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva. Estas fases son:

- **Inicio:** El objetivo de esta fase es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.
- **Elaboración:** El objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.
- **Construcción:** Durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.
- **Transición:** El sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación y aceptación y finalmente se despliega en los sistemas de producción.

La variación de la metodología AUP, denominada AUP-UCI se adapta al ciclo de vida definido para la actividad productiva de la UCI y es utilizada por el CEDIN en sus productos de software. El uso de esta técnica de modelado ágil permite encapsular los requisitos funcionales en Historias de Usuario (HU) o descripción de requisitos por procesos.(20)

Tabla 2 "Fases de AUP-UCI". (18)

Fases	**Objetivos**
Inicio	• Planeación del proyecto. • Estudio inicial de la organización cliente. • Alcance del proyecto. • Estimaciones de tiempo, esfuerzo y costo.
Ejecución	• Ejecución de las actividades requeridas para desarrollar el software. • Ajuste de los planes del proyecto. • Modelado del negocio. • Obtención de requisitos.

	• Se elaboran la arquitectura y el diseño. • Se implementa y se libera el producto.
Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

1.6 Conclusiones Parciales

- En este capítulo se realizó un esbozo de las principales características y conceptos importantes de los sistemas SCADA, haciendo énfasis en el módulo HMI.
- Se definieron y seleccionaron las tecnologías necesarias para llevar a cabo el cumplimiento del objetivo planteado en el trabajo.
- A partir de estos puntos se comenzará el desarrollo de la propuesta de solución.

CAPÍTULO 2. ANÁLISIS Y DISEÑO DE LA SOLUCIÓN

2.1 Introducción

El objetivo de este capítulo es explicar el análisis y diseño de la aplicación. Contiene la especificación de los requisitos funcionales y los no funcionales que debe cumplir el sistema, con sus pertinentes descripciones. Se define la arquitectura del sistema y los patrones de diseño utilizados.

2.2 Modelo de Dominio

Un modelo del dominio se utiliza con frecuencia como fuente de inspiración para el diseño de los objetos de software, muestra a los modeladores clases conceptuales significativas en un dominio del problema; es el artefacto más importante que se crea durante el análisis orientado a objetos. Es una representación de las clases conceptuales del mundo real, no de componentes de software(21).

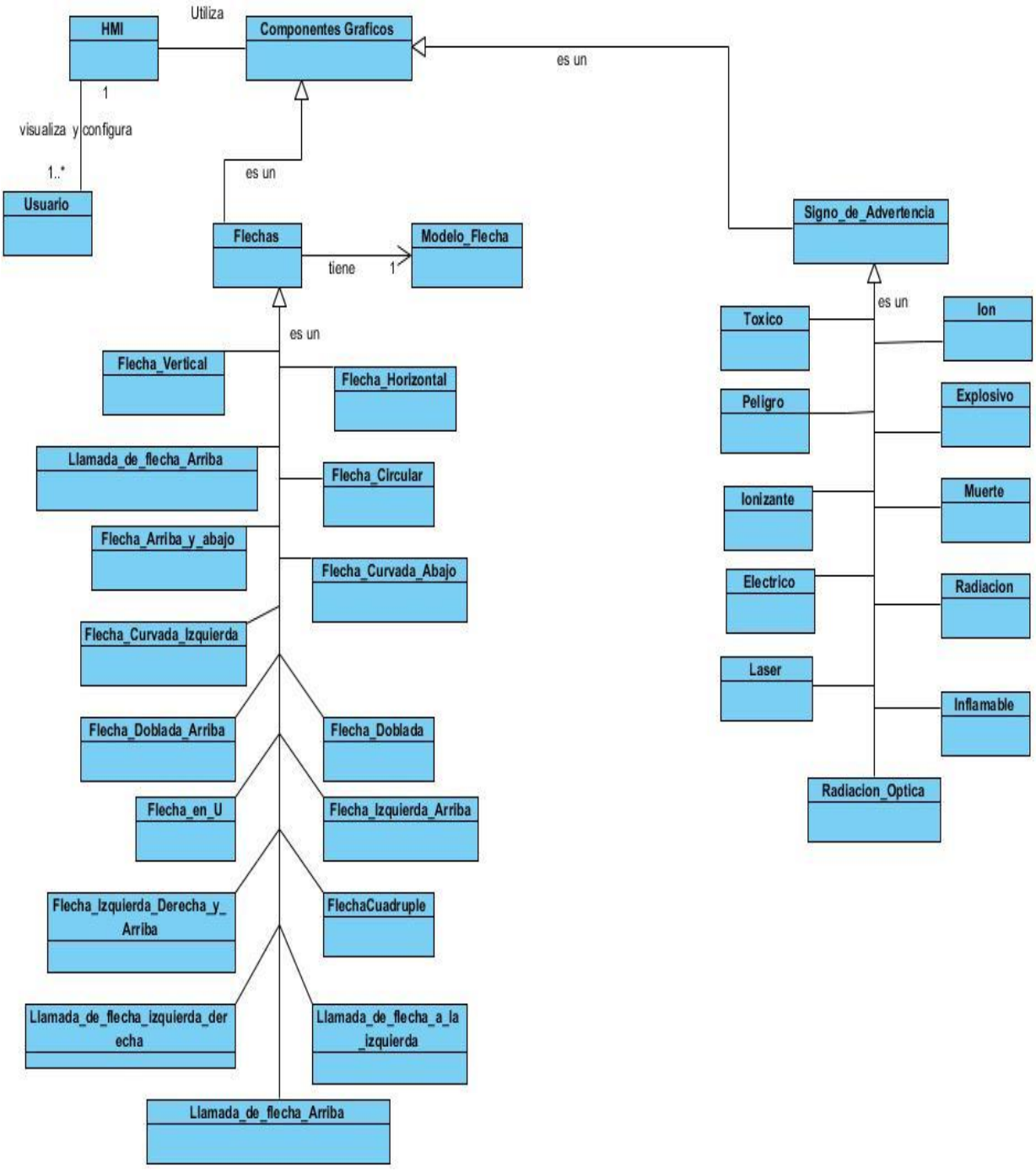


Figura 1"Modelo Dominio."

A continuación se describen las clases que participan en el negocio:

- **HMI:** es el módulo que se encarga de representar al usuario u operador lo que sucede en el proceso.
- **Componentes Gráficos:** representación abstracta de algún dispositivo del proceso que se está supervisando.

- **Flechas:** Las Flechas son Señales de Dirección que representan algún camino a función que se desea realizar ,refleja el status de la operación que se realiza ,para ello se utilizan varias flechas como :Flecha a la derecha con bandas, Flecha Horizontal, Flecha Vertical, Flecha Arriba y Abajo, Flecha Circular, Flecha Curvada Abajo, Flecha Curvada Izquierda, Flecha Doblada, Flecha Doblada Arriba, Flecha U, Flecha Izquierda y Arriba, Flecha Izquierda Derecha y Arriba, Flecha Cuádruple, Llamada de Flecha a la Izquierda, Llamada de Flecha Arriba, Llamada de Flecha Izquierda Derecha .
- **Signo de Advertencias:** Los Signos de Advertencia son componentes, que informan al usuario cual circunstancia o riesgo pueden ocurrir en el sistema, para esto se utilizan componentes gráficos como: Laser, Tóxico, Peligro, Ionizante, Eléctrico, Radiación Óptica, Inflamable, Ion, Explosivo, Muerte y Radiación.
- **Modelo Flecha:** Son las clases que guardan los datos que deben persistir de los componentes.

2.3 Especificación de Requisitos.

La ingeniería de requisitos es el conjunto de actividades implicadas en descubrir, documentar y mantener un conjunto de requisitos del software a desarrollar. La captura de los mismos es un proceso en el cual los datos son extraídos a partir de la aplicación de técnicas de captura de información como son la tormenta de ideas y las entrevistas (22).

Los requerimientos pueden dividirse en requerimientos funcionales y requerimientos no funcionales. Los requerimientos funcionales definen las funciones que el sistema será capaz de realizar. Describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Los requerimientos no funcionales tienen que ver con características que de una u otra forma puedan limitar el sistema, como por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad y estándares.

2.3.1 Requisitos Funcionales

Se determinaron los siguientes requisitos funcionales, comunes para ambos grupos de componentes:

RF1. El sistema debe permitir definir nombre del componente.

RF2. El sistema debe permitir configurar la opacidad del componente.

RF3. El sistema debe permitir configurar la dimensión de ancho del componente.

RF4. Configurar la dimensión de altura del componente.

RF5. Configurar la posición (X, Y) del componente.

RF6. El sistema debe permitir la rotación del componente.

RF7. El sistema debe permitir eliminar el componente

RF8. El sistema debe permitir añadir la animación de blinkeo al componente.

Requisitos Funcionales específicos para las Señales de Dirección.

RF9. El sistema debe permitir configurar color del borde del componente.

RF10. El sistema debe permitir configurar color de fondo del componente.

2.3.2 Requisitos no Funcionales

Un requisito no funcional o atributo de calidad es un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que estos corresponden a los requisitos funcionales, son todos los requisitos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento.(23)

Definen propiedades emergentes del sistema, tales como el tiempo de respuesta, la fiabilidad. (24)

Requerimientos de software

- El sistema debe funcionar en el Sistema Operativo Debian, en su versión 8.

Requerimientos de Hardware

Debe ser ejecutado en computadoras que tengan como requerimientos mínimos los siguientes:

- Microprocesador: dualcore a 1.5 GHz.
- RAM: 2 GB

Restricciones en el diseño y la implementación

- Se implementará utilizando el lenguaje de programación C++, en su versión 11.

- Se utilizará el IDE Qt Creator en su versión 2.5.0**.**
- Se empleará el marco de trabajo Qt, en su versión 5.3.1.

Requerimientos de usabilidad

- El sistema debe proporcionar una interfaz gráfica sencilla, intuitiva y fácil de entender.

Rendimiento

- En el ambiente de despliegue solo se admiten hasta 100 componentes.

2.4 Historias de Usuario

Las historias de usuario son una técnica utilizada en AUP-UCI para especificar los requisitos del software. Se realizó una por cada funcionalidad del sistema, fueron empleadas para hacer estimaciones de tiempo y para el plan de iteraciones. Cada historia de usuario que se muestra a continuación se definió lo más comprensible posible para que pueda ser implementada por el desarrollador en pocas semanas. (25).

2.4.1 Descripción Historias de Usuario

- Número: Posee el número asignado a la HU.
- Actor: El usuario del sistema que utiliza o protagoniza la HU.
- Nombre de HU: Atributo que contiene el nombre de la HU.
- Prioridad en Negocio: Evidencia el nivel de prioridad de la HU en el negocio.
- Puntos estimados: Este atributo es una estimación hecha por el equipo de desarrollo del tiempo de duración de la HU. Cuando el valor es 1 equivale a una semana ideal de trabajo.
- Iteración Asignada: Número de la iteración en la cual se desarrollará la HU.
- Descripción: Posee una breve descripción de lo que realizará la HU.
- Observaciones: Algunas aclaraciones que es importante señalar acerca de la HU.

Tabla 3 "Historia de Usuario" #1

Historia de Usuario	
Número: 1	**Nombre de Historia:** Definir nombre del objeto
Actor: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos estimados:** 0.5

Nivel de Complejidad: Alta	**Puntos reales:** 0.5
Descripción: El sistema proporciona una interfaz al cliente donde puede escribir el nombre del objeto, el mismo no debe exceder de 32 caracteres.	

Tabla 4 "Historia de usuario" #2

	Historia de Usuario
Número: 2	**Nombre de Historia:** Configurar la opacidad del componente.
Actor: Usuario	**Iteración Asignada:** 2
Prioridad en Negocio: Baja	**Puntos estimados:** 0.4
Nivel de Complejidad: Baja	**Puntos reales:** 0.4
Descripción: El operador al realizar clic izquierdo encima del componente y seleccionar la opción propiedades, en la parte derecha del editor se muestra un inspector de propiedades, en donde el operador puede seleccionar la transparencia que tendrá el componente representado en el rango de 0 a 100.	

Tabla 5 "Historia de usuario" #3

	Historia de Usuario
Número: 3	**Nombre de Historia:** Configurar la dimensión del ancho del componente.
Actor: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Medio	**Puntos estimados:** 0.6
Nivel de Complejidad: Medio	**Puntos reales:** 0.6
Descripción: El operador al realizar clic derecho encima del componente y seleccionar la opción propiedades, en la parte derecha del editor se muestra un inspector de propiedades, en donde el operador puede definirle el ancho que desea que tenga el componente. También el operador al marcar el componente con un clic, situar el puntero en el borde izquierdo o derecho del componente y presionar el clic izquierdo, puede aumentar o disminuir el ancho del componente a su preferencia.	

Tabla 6 "Historia de usuario" #4

Historia de Usuario	
Número: 4	**Nombre de Historia:** Configurar la dimensión de la altura del componente.
Actor: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Medio	**Puntos estimados**: 0.7
Nivel de Complejidad: Medio	**Puntos reales: :** *0.7*
Descripción: El sistema proporciona una interfaz al usuario donde el operador al dar clic derecho encima del componente y seleccionar la opción propiedades, en la parte derecha del editor se muestra un inspector de propiedades, en donde el operador puede definirle la altura que desea que tenga el componente. También el operador al marcar el componente con un clic, situar el puntero en el borde de arriba o debajo del componente y presionar el clic izquierdo, puede aumentar o disminuir la altura del componente a su preferencia.	

Tabla 7 "Historia de usuario" #5

Historia de Usuario	
Número: 5	**Nombre Historia:** Configurar la posición (X, Y) del componente.
Actor: Usuario	**Iteración Asignada:** 1
Prioridad: Alta	**Puntos Estimado:** *0.6*
Nivel de Complejidad: Alta	**Puntos Real:** 0.6
Descripción: El sistema proporciona una interfaz al usuario donde el operador al dar clic izquierdo encima del componente y seleccionar la opción propiedades, en la parte derecha del editor se muestra un inspector de propiedades, donde puede configurar la posición en la que se traslada el objeto sobre el eje X y el eje Y. También el operador puede seleccionar el componente dejando presionado el clic izquierdo sobre este y moverlo según su preferencia.	

Tabla 8 "Historia de usuario" #6

Historia de Usuario	
Número: 6	**Nombre de Historia:** Permitir la rotación del componente.
Actor: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Medio	**Puntos estimados:** 0.5
Nivel de Complejidad: Medio	**Puntos reales:** 0.5
Descripción: Establece un ángulo de rotación. Definir donde localizar el centro de rotación. El centro de rotación se puede localizar en distintas posiciones, en la parte inferior del widget, en la parte superior, en el centro, a la derecha o a la izquierda.	

Tabla 9 "Historia de usuario" #7

Historia de Usuario	
Número: 7	**Nombre de Historia:** Eliminar el componente.
Actor: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Medio	**Puntos estimados:** 0.6
Nivel de Complejidad: Media	**Puntos reales:** 0.6
Descripción: El sistema proporciona una interfaz al usuario donde el operador al dar clic izquierdo sobre el componente o al presionar la tecla Delete en el teclado da la opción de eliminar el componente.	

Tabla 10 "Historia de usuario" #8

Historia de Usuario	
Número: 8	**Nombre de Historia:** Añadir la animación de blinkeo.
Actor: Usuario	**Iteración Asignada:** 3
Prioridad en Negocio: Alta	**Puntos estimados:** 1

Nivel de Complejidad: Alta	**Puntos reales:** 1
Descripción: El sistema proporciona una interfaz donde el usuario selecciona el componente y escoge la animación de blinkeo.	

Tabla 11: "Historia de usuario" #9

	Historia de Usuario
Número: 9	**Nombre de Historia:** Configurar color del borde del componente.
Actor: Usuario	**Iteración Asignada:** 2
Prioridad en Negocio: Medio	**Puntos estimados:** 0.5
Nivel de Complejidad: Media	**Puntos reales:** 0.5
Descripción: El sistema proporciona una interfaz donde el usuario selecciona el componente y escoge el color que desea para pintar el borde del componente.	

Tabla 12: "Historia de usuario" #10

	Historia de Usuario
Número: 10	**Nombre de Historia:** Configurar color del fondo del componente.
Actor: Usuario	**Iteración Asignada:** 3
Prioridad en Negocio: Medio	**Puntos estimados:** 0.5
Nivel de Complejidad: Media	**Puntos reales:** 0.5
Descripción El sistema proporciona una interfaz donde el usuario selecciona el componente y escoge el color que desea para pintar el fondo del componente.	

2.5 Planificación del desarrollo.

Esta fase tiene como propósito establecer un acuerdo entre clientes y desarrolladores sobre el menor tiempo en que la mayor cantidad de historias de usuarios puedan ser utilizadas.

2.5.1 Estimación de esfuerzos e iteraciones por historia de usuario.

La siguiente tabla muestra la estimación de esfuerzo para cada una de las historias de usuarios definidas en el desarrollo de la solución propuesta, el plan de iteraciones realizado y la duración total de las mismas.

Tabla 13: "Estimación de esfuerzo"

Historias de Usuario	Puntos de Estimación	Iteración	Duración total de las iteraciones (semanas)
Definir nombre del objeto	0.5 semanas	1	5
Configurar la dimensión del ancho del componente	0.6 semanas		
Configurar la dimensión de la altura del componente	0.7 semanas		
Configurar la posición (X, Y) del componente.	0.6 semanas		
Permitir la Rotación del componente	0.5 semanas		
Eliminar el componente	0.6 semanas		
Configurar la opacidad del objeto del componente.	0.4 semanas	2	2

Configurar el color del borde del componente	0.5 semanas		
Configurar color del fondo del componente.	0.5 semanas		
Añadir la animación de blinkeo.	1 semanas	3	1

2.5.2 Plan de Iteraciones

Una vez definidas las historias de usuario (HU) se puntualiza la prioridad de cada una de ellas y la estimación de esfuerzo necesario para realizarlas para lo que se llevó a cabo un plan de iteraciones las cuales son mostradas a continuación.

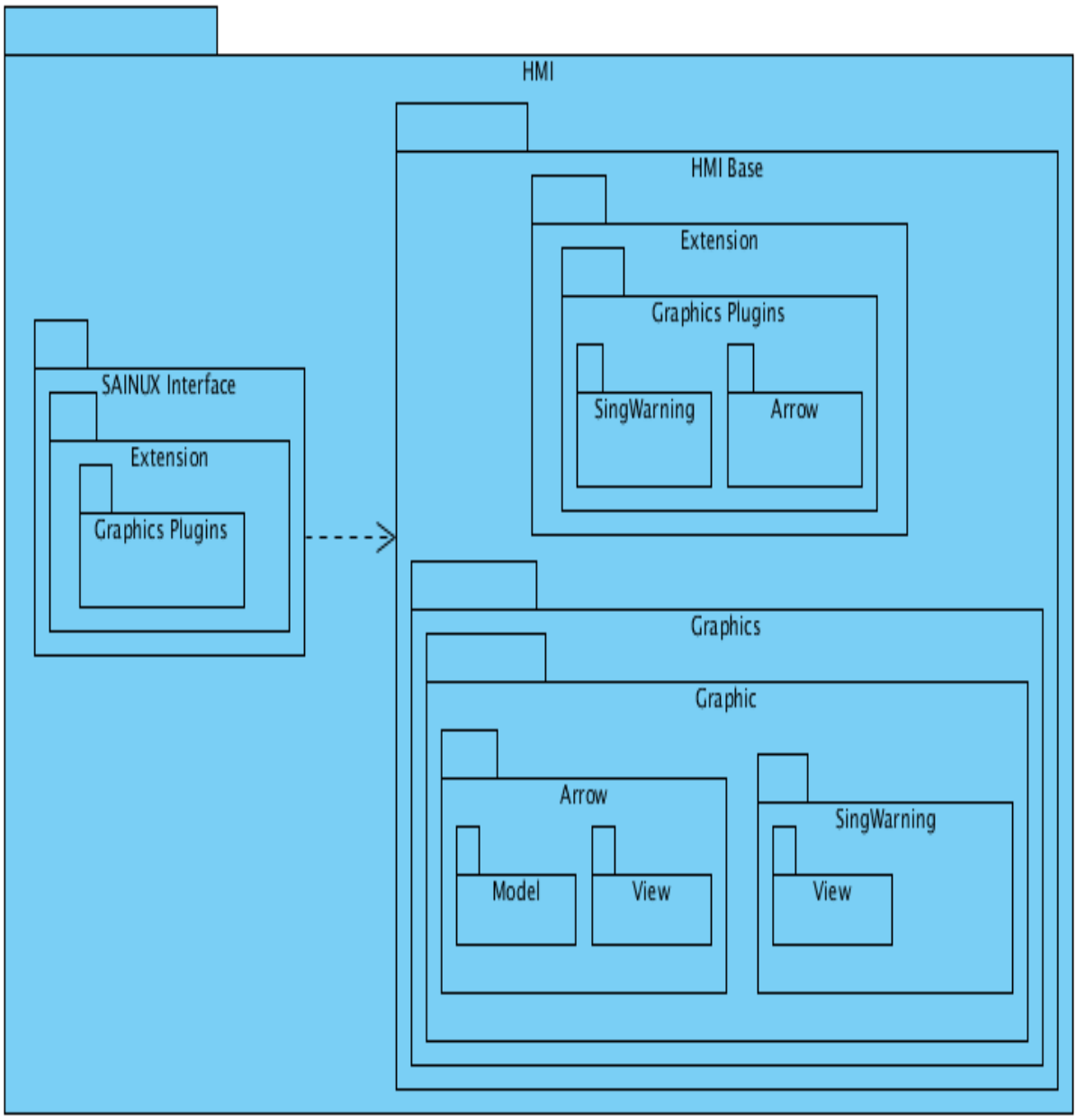
Se definieron Tres iteraciones para el desarrollo de las HU de la aplicación. A continuación, se explican y justifican cada una de las iteraciones:

- Iteración 1: Para esta iteración se desarrollan las HU 1, 3, 4, 5 ,6 ,7 las cuales se encargan de configuración de los componentes correspondientes, garantizando la representación gráfica de los parámetros de cada componente seleccionado.
- Iteración 2: En esta iteración se desarrollarán las HU 2 ,9 ,10 las cuales se encargan de configurar color (fondo y borde) y opacidad de los componentes.
- Iteración 3: En esta iteración se desarrolla las HU 8 la cual se encarga de realizar animaciones a cada componente gráfico.

2.6 Diagrama de Paquetes

Un diagrama de paquetes en el Lenguaje Unificado de Modelado representa las dependencias entre los paquetes que componen un modelo. Es decir, muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre las agrupaciones.

Figura 2: "Diagrama de Paquetes."



La solución está estructurada en dos partes; Componentes y Paletas de Componentes. La primera parte contiene la carpeta SAINUX Interface, esta contiene la subcarpeta Graphics Plugins que contiene las entidades referentes a las paletas de los componentes, pero de la interfaz SAINUX. En la segunda parte están las Paletas de Componentes divididas en la carpeta HMI Base donde se almacenan las propiedades comunes de cualquier módulo de un sistema SCADA SAINUX, este módulo contiene los paquetes Extensión y dentro de él la subcarpeta Graphics Plugins, donde se colocan las entidades relacionadas con las paletas Señales de Dirección y Signos de advertencia, y dentro de ellas según el tipo de componente que representa la paleta. El segundo paquete lo conforma Graphics donde se

guardan las entidades de la biblioteca de componentes gráficos, dentro de él la subcarpeta Graphic que agrupa las entidades que representan a los componentes Signos de Advertencia con sus entidades para las vistas y Señales de dirección con sus entidades para representar la Vista y el Modelo.

2.7 Diseño de Clases

El diseño es un proceso de resolución de problemas cuyo objetivo es encontrar y describir una forma. Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de agregación, ya que una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica; mostrando un conjunto de elementos que son estáticos, como las clases y tipos junto con sus contenidos y relaciones.

Figura 3: "Diagrama de clases de Símbolos de Advertencia."

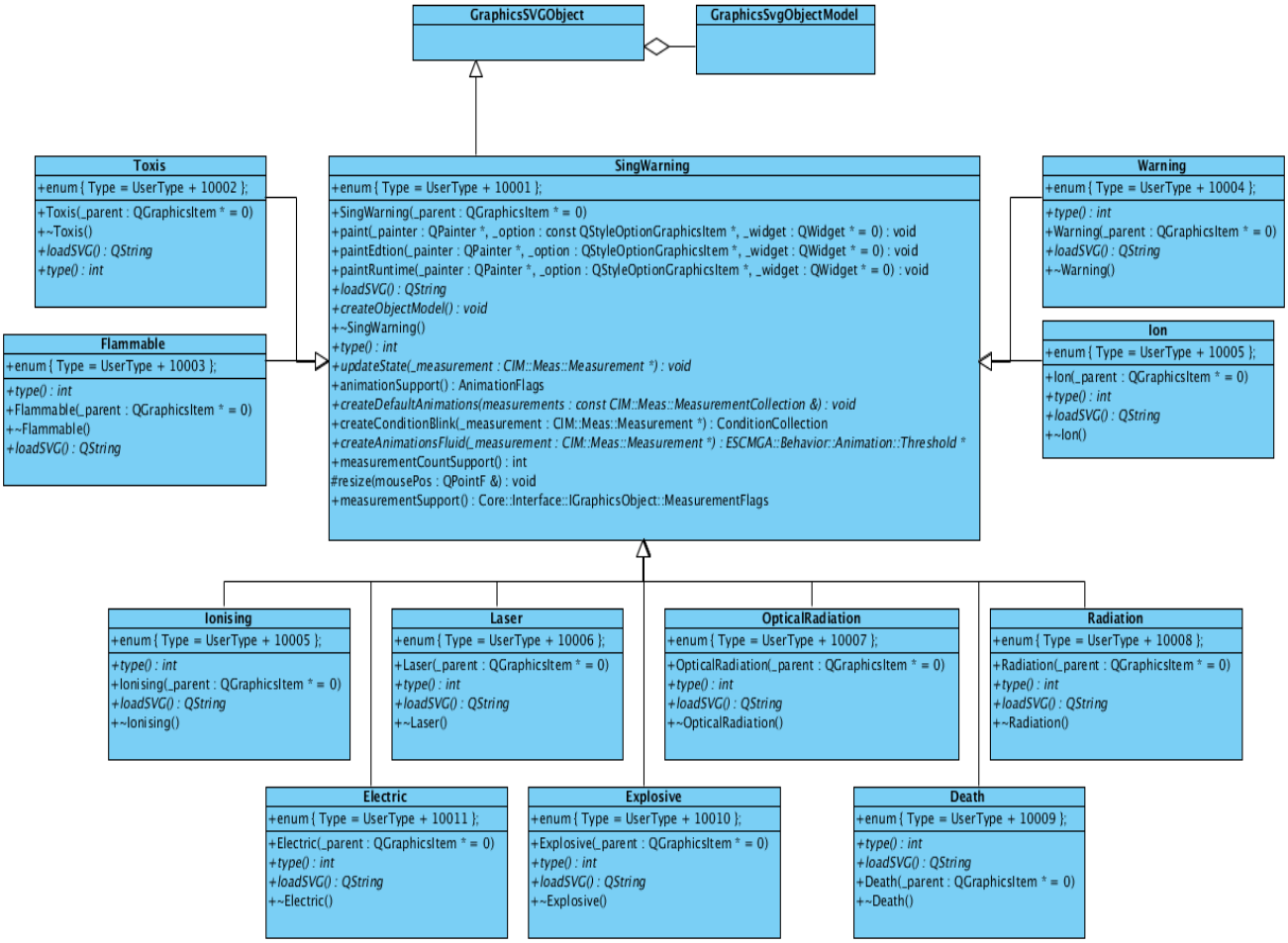
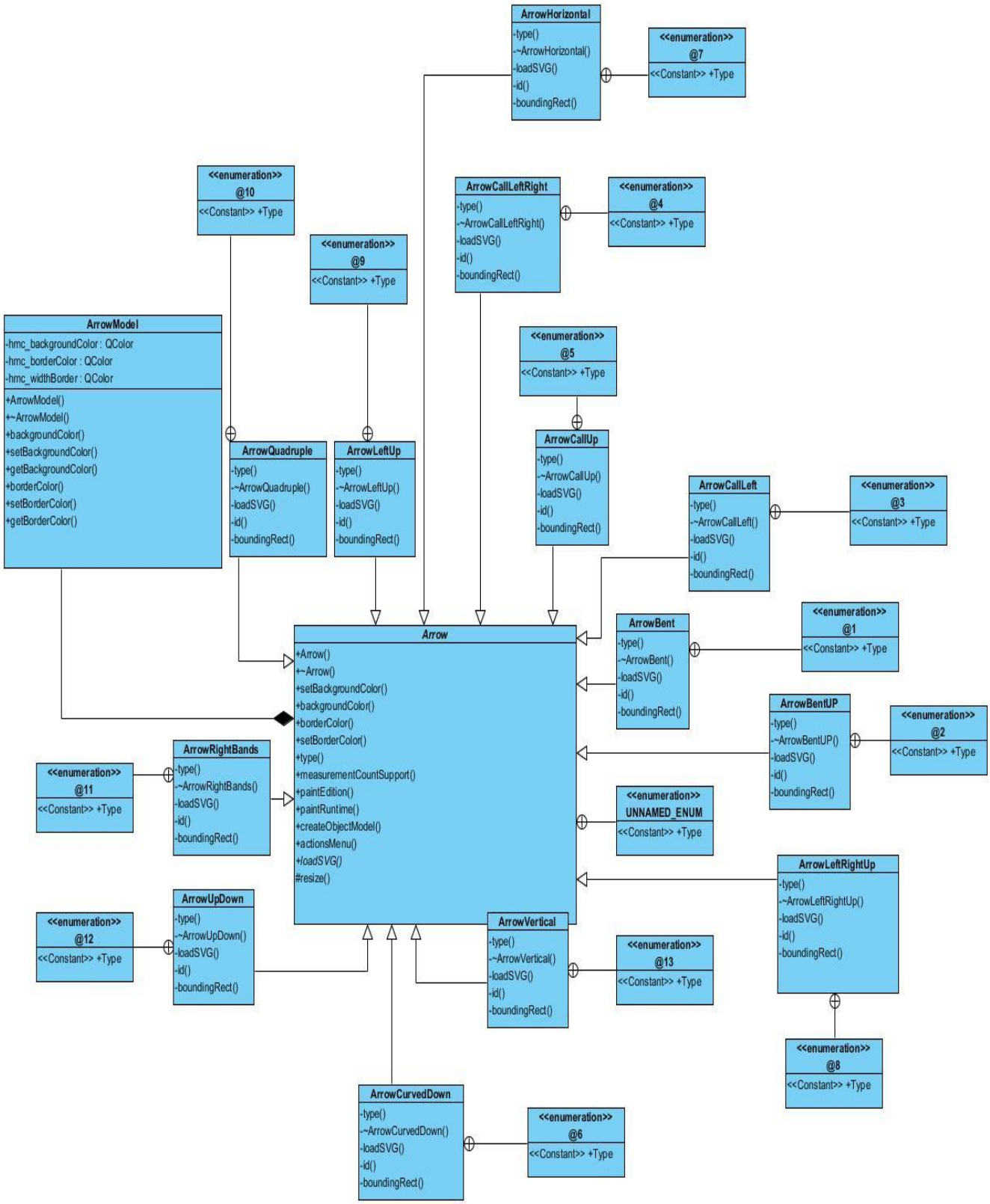


Figura 4: " Diagrama de Clases Señales de Dirección."



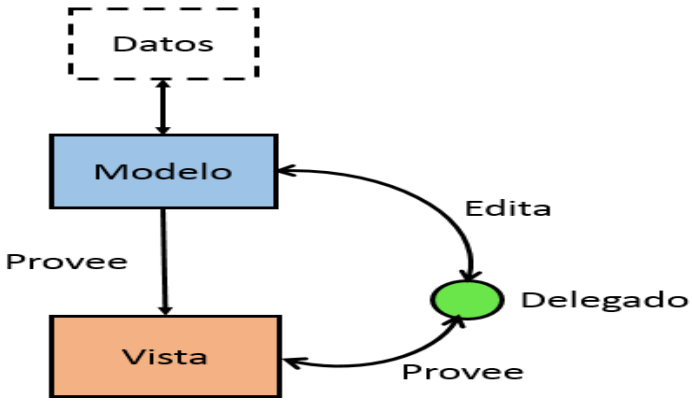
2.8 Patrones de Arquitectura

Los patrones de arquitectura son soluciones acertadas a un problema general, que establecen pautas para definir estructuras de diseño en el desarrollo de un software. Estos, están más enfocados a los diseños orientados a objetos pero según dichos patrones a menudo tienen en cuenta características de los objetos tales como la herencia y el polimorfismo que proporciona generalidad(26)**.**

El patrón de arquitectura usado es el Modelo / Vista que propone Qt donde el modelo comunica con una fuente de datos, proporcionando una interfaz para los otros componentes en la arquitectura. La naturaleza de la comunicación depende del tipo de fuente de datos, y la forma en que se implementa el modelo. La vista obtiene índices de modelo; estos son referencias a objetos de datos. Mediante el suministro de los índices del modelo, la vista puede recuperar los elementos de datos del origen de datos. En vistas estándar, un delegado hace que los elementos de datos, cuando se edita un elemento, el delegado se comunica con el modelo usando directamente los índices modelo(27).

En general, las clases de modelo / vista se pueden separar en los tres grupos descritos anteriormente: modelos, vistas y delegados. Cada uno de estos componentes se define por las clases abstractas que proporcionan interfaces comunes y en algunos casos, las implementaciones por defecto de características. Las clases abstractas están destinados a ser una subclase con el fin de proporcionar el conjunto completo de funcionalidad esperada por otros componentes; esto también permite que los componentes especializados puedan ser escritos.

Figura 5: "Patrón de Arquitectura Modelo/Vista."



De las clases que componen el Modelo se encuentran:

- Para el caso de las Flechas, se definió la clase ArrowModel.

De las clases que componen las vistas se encuentran:

- Para los Signos de Advertencia, se definieron las siguientes clases: Laser, OpticalRadiation, Explosive, Electric, Flammable.
- Para el caso de las Flechas se precisaron estas: Arrow, ArrowHorizontal y ArrowVertical.

2.9 Patrones de Diseño

Los patrones de diseño, tratan los problemas que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Se encargan de identificar clases, instancias, roles, colaboraciones entre estas, así como la distribución de responsabilidades. En resumen, es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular(28).
Estos patrones de diseño Banda de los Cuatro [Gang of Four, (GoF por sus siglas en inglés)] son clasificados principalmente por tres grupos como son(29).

- **Patrones de comportamiento:** se utilizan a la hora de definir como las clases y objetos interaccionan entre ellos.
- **Patrones creacionales:** utilizados para instanciar objetos, y así separar la implementación del cliente de la de los objetos que se utilizan. Con ellos intentamos separar la lógica de creación de objetos y encapsularla.
- **Patrones estructurales:** utilizados para crear clases u objetos incluidos dentro de estructuras más complejas.

Para la asignación general de responsabilidades en el software existen patrones denominados Patrones Generales de Asignación de Responsabilidades (GRASP, por sus siglas en inglés), que describen los principios fundamentales de asignación de responsabilidades a objetos, expresados como patrones. Seguidamente se exponen algunos patrones utilizados dentro del contexto de los módulos a implementar:

- **Alta cohesión:** Facilita la solución al problema ¿Cómo mantener manejable la complejidad? mediante la asignación de responsabilidades de manera que la información

que almacena una clase sea coherente y esté relacionada con la clase. El uso de este patrón se ve reflejado en todo el diagrama debido a que cada clase almacena la información de ella misma de manera coherente.

- **Experto**: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. De forma general en el diseño del sistema se basa en asignar a cada clase la responsabilidad que solo ellas pueden realizar, pues cada una cuenta con la información necesaria para llevarlas a cabo. Por lo que en todas las clases del sistema utilizan este patrón.

- **Bajo acoplamiento:** Facilita la solución al problema ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización? Reflejada en todo el diagrama.

- **Creador:** Permite asignar el responsable de la creación de una nueva instancia de alguna clase. Explica que clase es la encargada de crear objetos, en determinados escenarios de ejecución y guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito general de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento.

Los patrones GoF utilizados fueron Patrones de Comportamiento como:

- **Observador (Observer):** Define una dependencia de uno a muchos entre objetos de forma que, cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente. El uso de este patrón se ve reflejado en la funcionalidad ***updateState ()*,** la cual es utilizada para notificarle al componente que la animación asociada a él ha sufrido cambios o será configurada.

- **Plantilla (Template Method):** Define una operación, el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura. Es utilizado en clase abstracta donde el código común será usado por las clases que heredan de ella permitiendo la reescritura de determinados métodos. Reflejado en los métodos ***paintRuntime ()***, ***loadSVG ()*** y ***paintEdition ()***.

2.10 Conclusiones Parciales

En este capítulo se trataron temas referentes al análisis y diseño del componente para la representación de Signos de advertencia y Señales de dirección, lo que permitió arribar a las siguientes conclusiones:

- La modelación del dominio permitió establecer un punto de partida para lograr un correcto diseño del sistema.
- Se especificaron además los requisitos del sistema que permitieron identificar las funcionalidades con las que este contará y que darán respuesta a las necesidades del usuario.
- Se estableció la arquitectura que tendrá el componente a desarrollar empleándose el patrón arquitectónico modelo-vista.
- Con la realización del diagrama de clases, se obtuvo una visión más clara del sistema en términos de implementación.

CAPÍTULO 3. IMPLEMENTACIÓN Y PRUEBAS

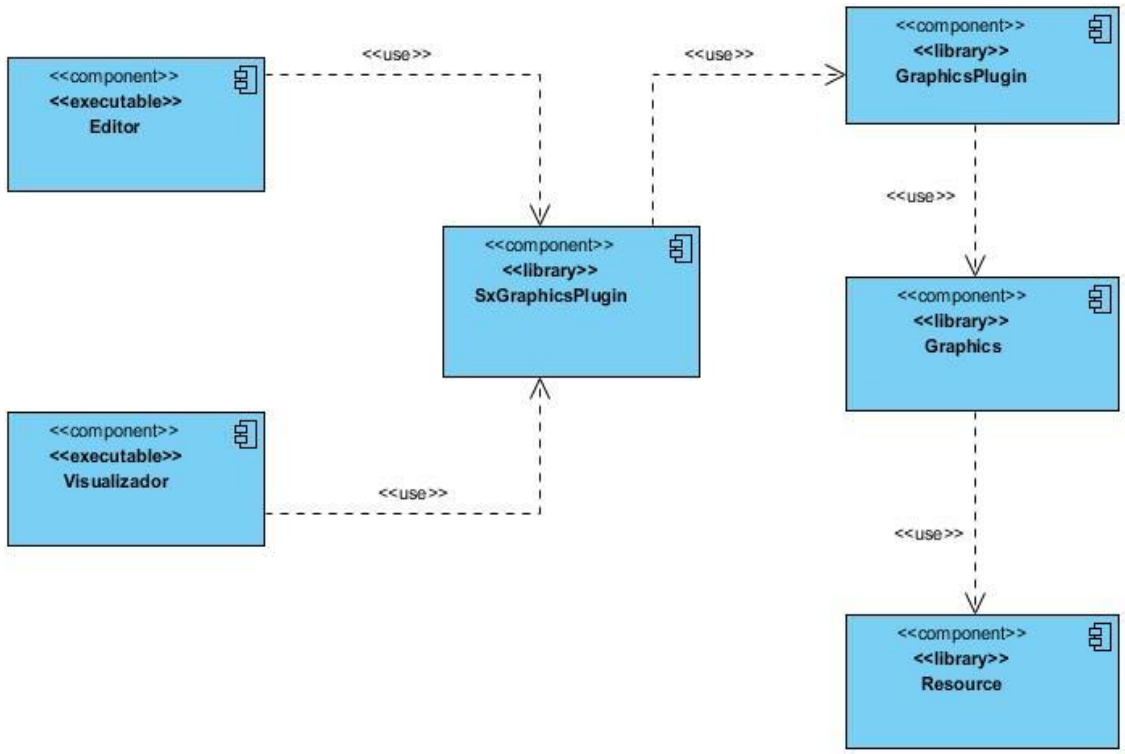
3.1 Introducción

En este capítulo se expondrá el estándar de codificación utilizado para el desarrollo de la solución, así como su respectivo diagrama de componentes y de despliegue. Luego de haber realizado el diseño se desarrolla la solución y luego se somete la herramienta a pruebas internas para detectar posibles fallas, mediante las pruebas de aceptación donde se validarán las soluciones. Una vez corregidos los errores podrán ser desplegados para su utilización.

3.2 Diagrama de Componentes

Un diagrama de componentes representa la separación de un sistema de software en componentes físicos (por ejemplo archivos, módulos, paquetes, base de datos, código fuente, binario o ejecutable) y muestra las dependencias y organización existente entre estos componentes. Los diagramas de componentes prevalecen en el campo de la arquitectura de software, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.

Figura 6: "Diagrama de Componentes."



El diagrama de componentes antes dibujado está estructurado por ejecutables y bibliotecas:

Editor: Es la aplicación donde se trabaja en el entorno de configuración para representar los procesos del campo.

Visualizador: Es una aplicación donde se supervisa el área configurada para interactuar con los componentes gráficos.

Biblioteca Graphics Plugins: En esta biblioteca se almacenan los Plugins de los objetos gráficos.

Biblioteca Graphics: Aquí están almacenados objetos gráficos que se utilizan para representar lo componentes gráficos, ya sea de forma simple o componentes complejos.

SxGraphics Plugins: Es una interfaz de la biblioteca Graphics Plugins específica para los componentes de SAINUX2.0.

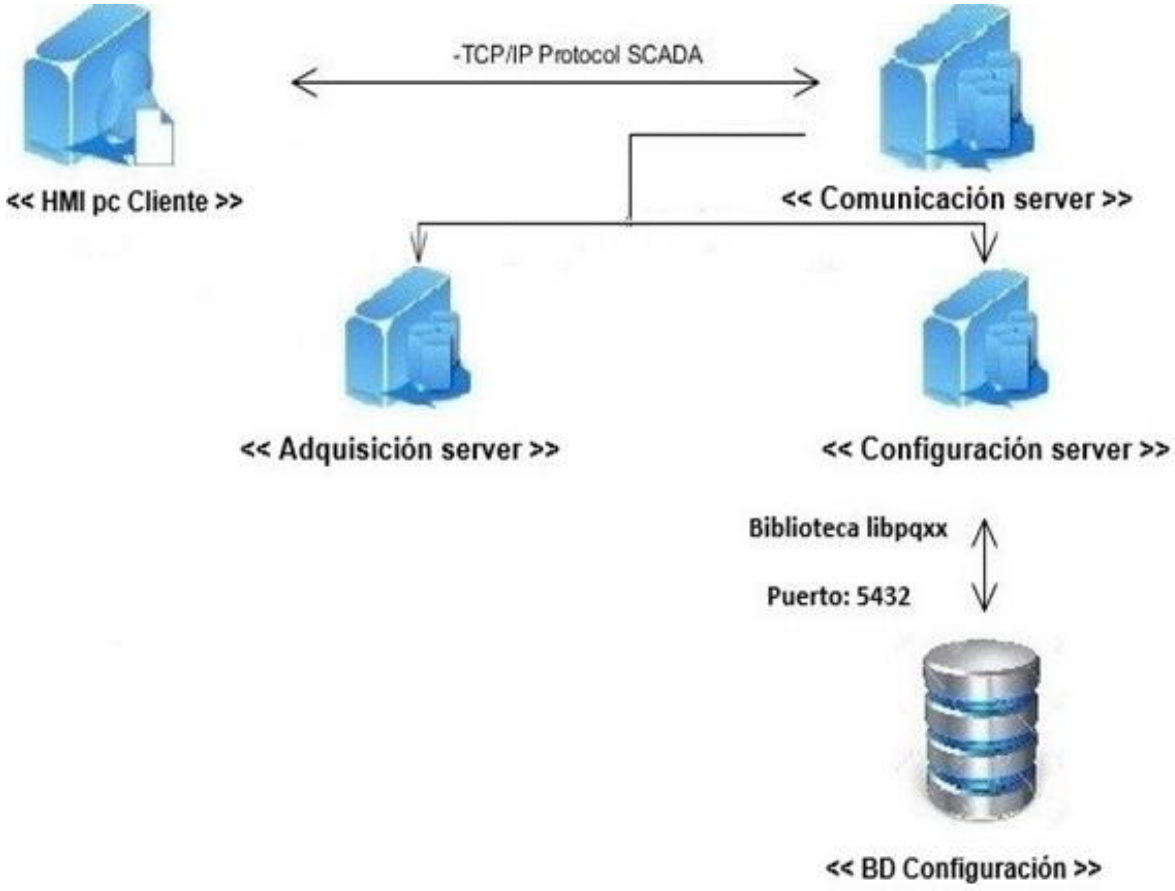
Resource: Biblioteca donde se encuentran agrupadas cada una de las entidades encargadas de brindar las imágenes, los iconos y los diseños de los componentes en SVG.

3.3 Diagrama de Despliegue

Con el objetivo de proveer una descripción de la distribución física del sistema se realiza el modelo de despliegue, mediante el cual se muestran las relaciones físicas de los distintos nodos que componen el sistema.

Los diagramas de despliegue son capaces de describir la arquitectura física del sistema durante la ejecución en términos de procesadores, dispositivos y componentes de software. Permite una mejor comprensión entre la correspondencia de la arquitectura de software y la arquitectura de hardware.

Figura 7: "Diagrama de Despliegue."



3.3.1 Descripción del diagrama de despliegue

La PC-Cliente representa el ordenador donde se instala el módulo HMI, en el cual pueden estar instalados los dos entornos o simplemente uno de ellos, es en este módulo específicamente donde se encuentra la solución desarrollada. Los módulos adquisición, configuración y base de datos históricos se ejecutan en una PC, que puede ser un servidor para cada uno por separado poniendo en evidencia la arquitectura distribuida del SAINUX, la conexión entre estos módulos y la PC-Cliente se realiza utilizando comunicación TCP/IP con el protocolo SCADA desarrollado por el centro. El módulo de configuración y base de datos históricos están conectados a sus respectivas bases de datos por la biblioteca libpqxx usando el puerto 5432.

3.4 Estándar de Codificación

Uno de los instrumentos que facilitan el trabajo y la calidad del software son la adopción de estilos y estándares de codificación. El uso de estos estándares tiene innumerables ventajas entre ellas lograr un estilo de código homogéneo asegurando su legibilidad y proveer una guía para el encargado del mantenimiento/actualización del sistema, con código claro y bien documentado. Además ayuda a mejorar el proceso de codificación haciéndolo en gran medida eficiente y en muchos casos reutilizables.

La solución propuesta en este trabajo es parte del sistema SCADA SAINUX 2.0 el estándar de codificación utilizado fue definido por el proyecto. Algunas de las pautas definidas por el proyecto son:

- Es importante especificar el nombre del autor y la fecha de creación de cualquier estructura de código, para ello se utilizan los comandos @autor y @date.
- El código será escrito en inglés y la documentación en español.
- Las variables y funciones comienzan con letra minúscula. Cada palabra consecutiva en el nombre comienza con letra mayúscula.
- Las variables de corto alcance, por ejemplo un contador, deberían tener nombres cortos y simples
- Los atributos de las clases deben empezar con m_ seguido del nombre del atributo, en el caso de atributos compuestos, la inicial de la segunda palabra debe comenzar con mayúscula.
- Las funciones utilizan la nomenclatura camello.
- Los parámetros que recibe una función debe empezar con _ (guion bajo).
- Todas las funcionalidades y atributos deben seguir el siguiente formato: para documentar @brief Nombre del método.
- Ninguna función debe tener más de 200 líneas.
- Los valores de los numerativos deben ser con letras mayúsculas.
- Las secciones public, protected y private son declaradas en el orden expuesto.

3.5 Solución del Problema

En este epígrafe se muestran la interface del Sistema SCADA SAINUX 2.0, donde se encuentran las bibliotecas de componentes del HMI en el entorno de configuración (Editor) y el entorno de visualización (Runtime).

3.5.1 Despliegues del sistema

La biblioteca de componentes gráficos del HMI SAINUX 2.0 no le provee al mantenedor componentes para la representación de Señales de dirección y Signos de Advertencia. Para solventar dicha situación el mantenedor utiliza o combina varias figuras geométricas simples como: polilínea, curvas, rombos, semicírculos y líneas. En la siguiente figura se muestran los accesorios que se utilizaban anteriormente.

Figura 8 "Componentes Básicos del sistema."

En la siguiente imagen vemos los accesorios para la representación de las Señales de dirección, al añadir un total de 16 Señales de dirección teniendo estas nuevas funcionalidades, vemos cómo se va dando cumplimiento a los problemas planteados sobre la representación de figuras simples en el sistema.

Figura 9 "Componentes Señales de Dirección del sistema."

Al añadirle al sistema un total de 11 componentes gráfico para la representación de Signos de Advertencia, el mantenedor puede brindarle la seguridad al usuario e informar mediante señales simples, bajo cual riesgo o circunstancia puede estar y este tomar las medidas necesarias para evitar la pérdida de vidas humanas. En la imagen que a continuación se les muestra vemos los accesorios para la representación de Señales de advertencia en el entorno de configuración del sistema.

Figura 10 "Componentes Signos de Advertencia del sistema."

En las imágenes siguientes vemos algunos despliegues, donde podemos observar la biblioteca de componentes del HMI en la interfaz del sistema. La región que se encuentra subrayada en azul se ve reflejado el inspector de propiedades donde se encuentran las funcionalidades que deben ser configuradas en dichos componentes a la hora de realizar un despliegue; en la región verde se encuentra el área de despliegue donde el componente se va adaptando a la configuración dada por el usuario.

Figura 11 "Interfaz del Sistema en el entorno de configuración."

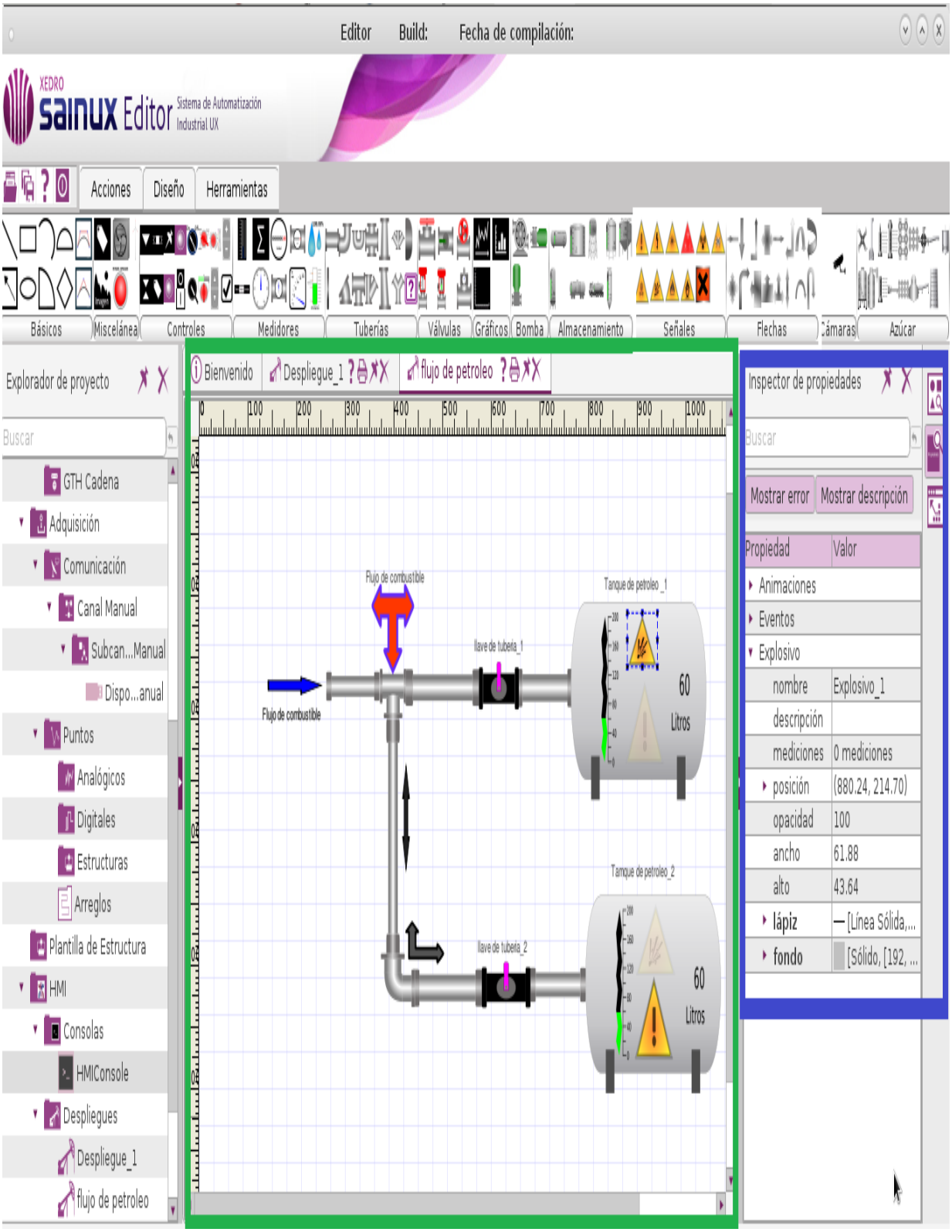
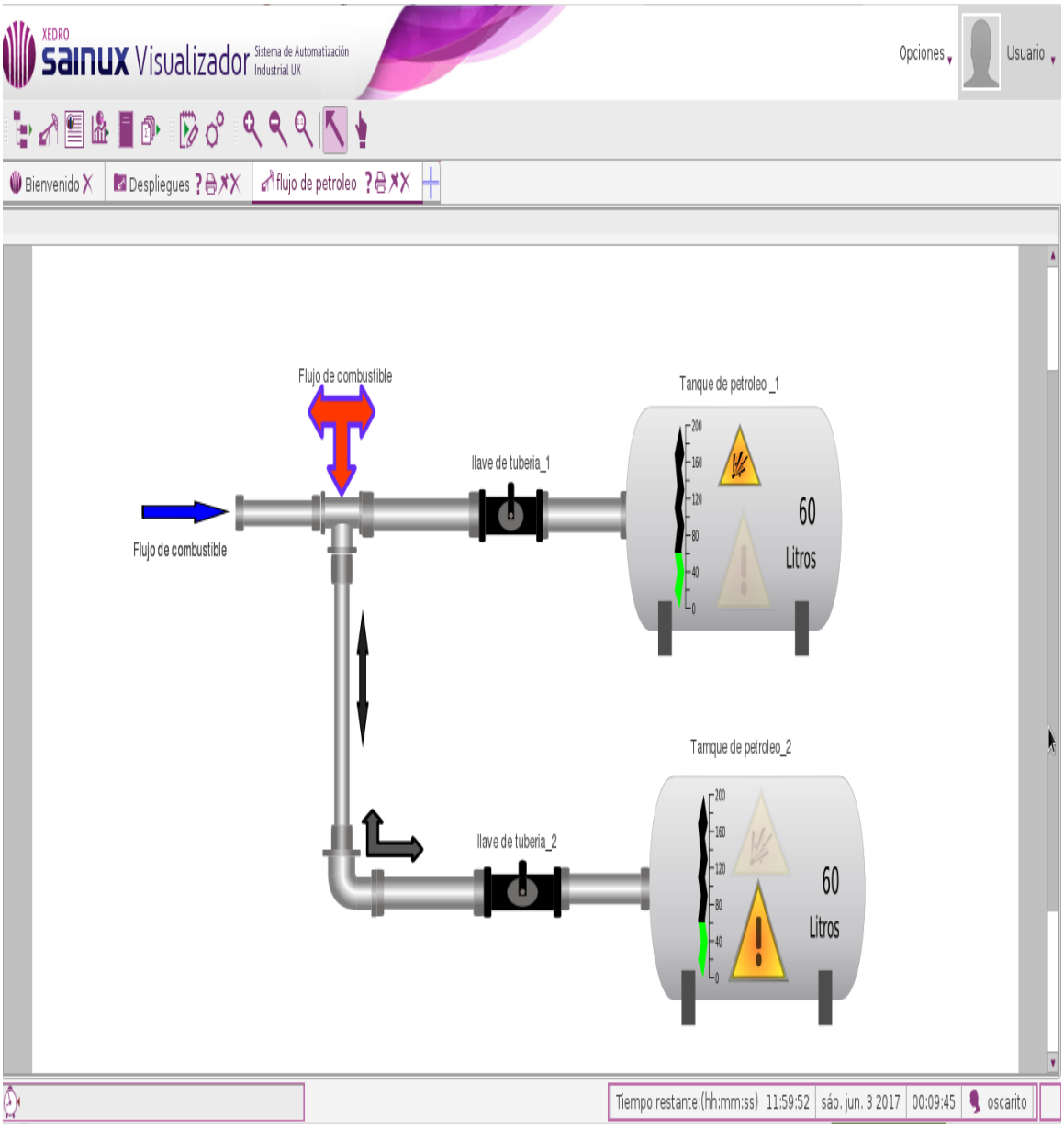


Figura 12 "Interfaz del Sistema en el entorno de Visualización."



3.6 Pruebas

La prueba del software es un elemento crítico para la garantía de la calidad del software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Además, esta etapa implica:

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.

- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

La prueba es un proceso que se enfoca sobre la lógica interna del software y las funciones externas. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Estas son un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente. Por esta razón, se deben definir en el proceso de la ingeniería del software.

Se decide realizar las pruebas de aceptación a los módulos implementados, debido a que el objetivo de estas, es verificar que el sistema cumpla con los requisitos establecidos por el usuario. De esta forma se puede obtener el grado de satisfacción del cliente.

3.6.1 Tipos de Pruebas

- **Pruebas Unitarias:** Tiene como objetivo: Asegurar que el código funcione de acuerdo con las especificaciones y que el módulo lógico sea válido. Esta prueba se describe de la siguiente forma, se particionan los módulos en unidades lógicas fáciles de probar. Por cada unidad hay que definir los casos de prueba (pruebas de caja blanca). Para esto los casos de prueba deben diseñarse de forma tal que se recorran todos los caminos de ejecución posibles dentro del código bajo prueba; por lo tanto el diseñador debe construirlos con acceso al código fuente de la unidad a probar. Los aspectos a considerar son los siguientes: Rutinas de excepción, Rutinas de error, Manejo de parámetros, Validaciones, Valores válidos, Valores límites, Rangos, Mensajes posibles.(30)
- **Prueba de Integración:** Tiene como objetivo: Identificar errores introducidos por la combinación de programas probados unitariamente. Determinar cómo la base de datos de prueba será cargada. Verificar que las interfaces entre las entidades externas (usuarios) y las aplicaciones funcionan correctamente. Descripción: Describe cómo verificar que las interfaces entre las componentes de software funcionan correctamente. Decide qué acciones tomar cuando se descubren problemas(31).

- **Prueba de Regresión:** Identificar errores introducidos por la combinación de programas probados unitariamente. Determina como la base de datos de prueba será cargada(31).
- **Pruebas de Seguridad y Control de Acceso:** Tiene como objetivo verificar que un actor solo pueda acceder a las funciones y datos que su usuario tiene permitido. Nivel de Seguridad del Sistema: Verificar que solo los actores con acceso al sistema y a la aplicación están habilitados para accederla. Descripción: Las pruebas de seguridad y control de acceso se centran en dos áreas claves de seguridad(31).
- **Pruebas de aceptación**: Verifican que el sistema que recibe funciona y lo hace de acuerdo con las especificaciones. Las pruebas de aceptación se realizan de acuerdo con protocolos específicos del producto. Al participar en un proceso de prueba de aceptación, se puede obtener un conocimiento más profundo acerca de cómo funciona el equipo. Son realizadas principalmente por los usuarios con el apoyo del equipo del proyecto. El propósito es confirmar que el sistema está terminado, que desarrolla puntualmente las necesidades de la organización y que es aceptado por los usuarios finales(32).

Una vez descritas las pruebas anteriores se llega a la conclusión que las pruebas de aceptación son las indicadas para evaluar el software, porque es la prueba planificada y organizada formalmente para determinar si se cumplen los requisitos de aceptación marcados por el cliente. Sus características principales son las siguientes: Estas pruebas derivan de las HU que se han implementado como parte de la liberación del software. Una prueba de aceptación es como una caja negra. Cada una de ellas representa una salida esperada del sistema. Es responsabilidad del cliente verificar la corrección y toma de decisiones acerca de estas pruebas.

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

- **Código:** Representa al caso de prueba, incluye el número de HU, de la prueba y si posee diferentes escenarios.
- **HU:** Número de la HU a la cual pertenece.
- **Nombre:** Junto al código conforma el identificador del caso de prueba.
- **Descripción:** Acción que debe realizar el sistema.

- **Condiciones de ejecución:** Describe las características y elementos que debe contener el sistema para realizar el caso de prueba.
- **Entrada/Pasos de Ejecución:** Incluye las entradas necesarias para realizar el sistema, además de los pasos para realizar el caso de prueba.
- **Resultados Esperados:** Descripción de la respuesta del sistema ante el caso de prueba.
- **Evaluación de la prueba:** Clasificación de la prueba en satisfactoria o insatisfactoria

La siguiente tabla muestra las pruebas de aceptación de la HU1 perteneciente a la primera iteración de sistema

Tabla 14: "Caso de Prueba Aceptación #1."

Caso de Prueba Aceptación	
Numero: 1	**Historia de usuario: 1**
Nombre: Definir nombre del componente.	
Descripción: Comprobar que el componente se nombra de forma correcta sin utilizar números o símbolos.	
Condiciones de Ejecución: El usuario debe comprobar que se puede configurar el nombre del componente.	
Entradas/ Pasos de Ejecución: - Seleccionar el componente. - Clic derecho con el ratón. - Seleccionar la opción Propiedades. - Cambiar el nombre.	
Resultado esperado: El nombre ha sido cambiado.	

Evaluación de la prueba: Prueba satisfactoria.

La descripción de las siguientes pruebas de aceptación se encuentra en el Anexo 1.

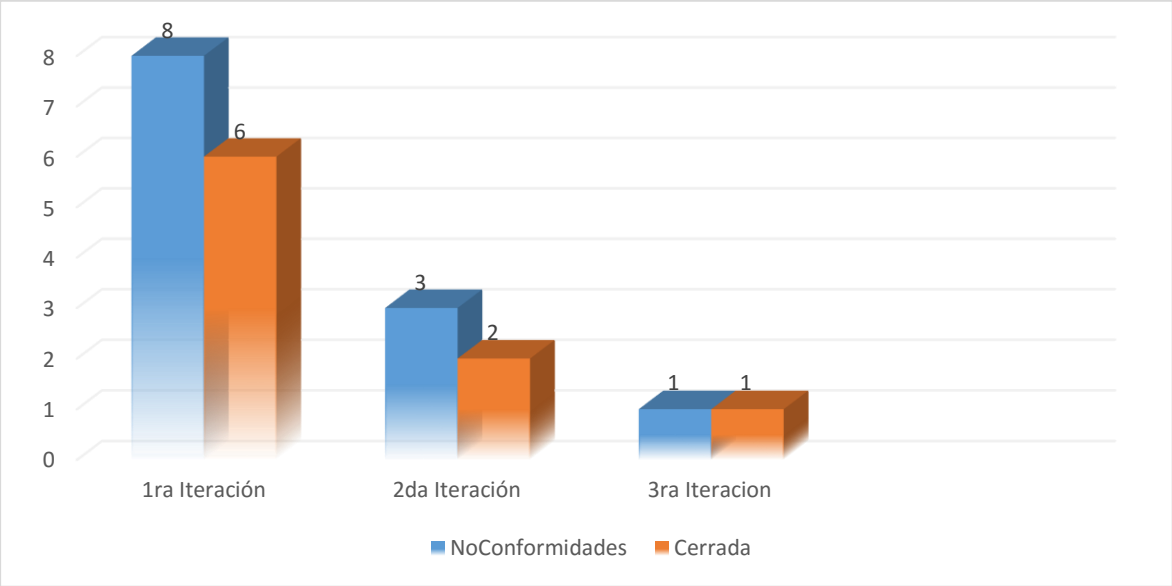
3.6.2 Diseño de Casos de Prueba

Tabla 15: "Diseño de Caso Prueba."

Pruebas del Sistema	HU	Iteración	NC	Cerrada
	10	1ra	8	6
		2da	3	2
		3ra	1	1

Una vez realizados los casos de pruebas para un total de 10 historias de usuarios, con tres iteraciones se eliminaron las 12 no conformidades encontradas.

Figura 13 "Plan de Iteraciones."



Como resultado se detectó que los errores encontrados pueden clasificarse según su tipo:

- Para una cantidad de 8 no conformidades se revelan errores de presentación porque no se visualiza correctamente toda la información que se desea mostrar.
- Para las siguientes 4 no conformidades los errores fueron de funcionalidades que no realizaban la acción prevista y errores visuales donde el sistema no mostraba lo requerido.

3.7 Conclusiones Parciales

En el presente capítulo se realizó una descripción de la implementación y las pruebas realizadas al sistema. En el mismo se generaron los artefactos necesarios para la implementación y las pruebas del sistema, por lo que se puede arribar a las siguientes conclusiones:

- La realización del diagrama de componentes permitió entender la relación entre los componentes que conforman el sistema.
- Mediante las pruebas ejecutadas al sistema, se lograron encontrar errores funcionales en el sistema.
- Obteniendo resultados satisfactorios en la última iteración de pruebas se puede concluir que el sistema no presenta errores funcionales.

.

CONCLUSIONES GENERALES

Con la realización de este trabajo se desarrollaron componentes gráficos para la representación de Signos de advertencia y Señales de dirección sobre la Interfaz Hombre Máquina del sistema SCADA SAINUX 2.0. De esta forma se le da cumplimiento al objetivo propuesto al inicio de la investigación, además se comprobó que:

- La metodología de desarrollo de *software* utilizada, permitió la correcta definición de las Historias de Usuario, logrando así ver con claridad los requisitos funcionales.
- Con la implementación de la solución, Componente gráfico para la representación de Signos de advertencia y Señales de dirección en el HMI del SCADA SAINUX 2.0, se eliminan los problemas descritos en la situación problemática
- Mediante la realización de las pruebas de aceptación se comprobó que la propuesta de solución cumple con los requerimientos pactados con el Centro.

RECOMENDACIONES

Se recomienda seguir ampliando la gama de componentes gráficos que representan signos de advertencia y señales de dirección.

Incorporar los componentes gráficos en el visor Web que está desarrollado en el Centro CEDIN.

REFERENCIAS BIBLIOGRÁFICAS

1. DAGOBERTO MONTEROS, DAVID B. BARRANTES Y JOSÉ M. QUIRÓS. *Introducción a los sistemas de control, supervisión y adquisición de datos (SCADA)*. 2004.

2. DAYRA IRIS HECHAVARRÍA RODRÍGUEZ, LANNIE OCTAVIO HERRERA PÉREZ. Procedimientos para la Gestión de Requisitos en el SCADA SAINUX. . Octubre 2012. Vol. Vol. 6. No. 4,.

3. Introducción a HMI. In: Universidad Nacional de Quilmes, [no date]. p. Págs. 1, 2.

4. ADOLFO YASSER SANTANA ROJAS. *Visualizador web de la Interfaz Hombre-Máquina del SCADA Guardián del ALBA*. La Habana: UCI, 2015.

5. *Señales y Símbolos de seguridad*. INSTITUTO ECUATORIANO DE NORMALIZACIÓN. Primera edición

6. Señales de Seguridad Tipos, Colocación y Formas. [Online]. [Accessed 21 January 2017]. Available from: http://www.areatecnologia.com/se%C3%B1ales-seguridad.htm

7. Definición de Señal. [Online]. [Accessed 21 January 2017]. Available from: http://conceptodefinicion.de/senal/

8. Qué es SVG. [Online]. [Accessed 21 January 2017]. Available from: http://aprendeweb.net/NT/svg/svg_1.php

9. BOBADILLA, Daniel. Inkscape: software libre para diseño vectorial • Freepress Coop. *Freepress S. Coop. Mad.* [online]. 24 March 2011. [Accessed 21 January 2017]. Available from: http://www.freepress.coop/inkscape-software-libre-para-diseno-vectorial/Descubre nuestro blog de noticias y recursos para el diseño y la comunicación.

10. Debian -- El sistema operativo universal. [Online]. [Accessed 21 January 2017]. Available from: http://www.debian.org/

11. Florentino, Deiry. Historia Del Computador, Arquitectura, Lenguajes de Programación y Algoritmos - reparacion de computadoras. [Online]. [Accessed 21 January 2017]. Available from: http://nectacellcomputer.jimdo.com/historia-del-computador-arquitectura-lenguajes-de-programación-y-algoritmos/Esto se llama Cultura General

12. Importancia de los Framework. [Online]. Available from: http://qt.nokia.com/products/developer-tools/.

13. Raydel Raúl Viñoso Sosa, Alexander Roquero Figuer. Sistema Gestor de Proceso de Media v2. [Online]. De diciembre de 2015. Available from: http://publicaciones.uci.cu/index.php/SC

14. Visual Paradigm. [Online]. [Accessed 22 January 2017]. Available from: https://www.visual-paradigm.com/features/

15. Modelos Y Metodologías Para El Desarrollo De Software. [Online]. [Accessed 22 January 2017]. Available from: http://www.eumed.net/tesis-doctorales/2014/jlcv/software.htm

16. BOOCH, JAMES. *El Proceso Unificado de Desarrollo de Software*. I. 2000.

17. *Pressman*. [No date]. Pressman, Cap. 04, Desarrollo Ágil

18. FRANCISCO MORA MARI ISABEL and JOSÉ PASCUAL. *Sommerville_Parte_I_Visión_General.pdf* [Online]. 7ma. Madrid, 2005.

19. Luis Orlando. *Tesis Luis Orlando Batista. REV. 01-14.pdf*. Habana, 2016.

20. TAMARA RODRÍGUEZ SÁNCHEZ. *Metodología de Desarrollo para la Actividad Productiva de la UCI* [Online]. Available from: http://excriba.prod.uci.cu/page/context/shared/sharedfiles/Metodologiauci.pdf

21. YORDANIS BRIDÓN. *Interfaz asíncrona para la comunicación de los controladores lógicos programables utilizando el protocolo industrial Ethernet/IP. 2008.)*. 2008. Danger

22. CAROLINA MARTINEZ. *Ingeniería de Software 1*. 2012.

23. PRESSMAN. *Software Engieering*. 2001.

24. PEARSON, Sommerville. *Ingeniería Del Software*. Vol 7. 2005.

25. BECK, KENT, ANDRES, YCYNTHIA. *Extreme Programming Explained: Embrace Change*. S.l.: U. S.

26. *Ingeniería de Software (Parte IV Desarrollo)*. 7ma. s.l.: PEARSON Addison-Wesley, 2005. ISBN 84-7829-074-5.

27. Qt Documentation. [Online]. Available from: http://doc.qt.io/qt-4.8/model-view-programming.html.

28. LARMAN. *UML y Patrones*. 2da Edición. 2002.

29. Available from: http://programacionsolida.com.ar/2012/07/patrones-de-diseno-de-comportamiento_09.html

30. GALIPENSO, Antini Botía Martínez María Isabel Alfonso and JOSE PASCUAL TRIGUEROS JOVER. *Sommerville Parte V, Verificación y Validación.pdf*. Madrid 2005: ISBM: 84-7829-074-5, [no date].

31. EliuMM. In: [online]. 5 May 217AD. Available from: http://javablog.eliumontoya.com/home/tipodepruebasparadesarrollodesoftware.

32. Niveles de Pruebas de un software. [Online]. 1 May 2017. Available from: www.ecured.cu

ANEXO 1 PRUEBAS DE ACEPTACIÓN

Tabla 16: "Caso de Prueba Aceptación #2."

Caso de Prueba Aceptación	
Número: 2	**Historia de usuario: 2**
Nombre: Configurar la opacidad del componente.	
Descripción: Comprobar que al componente puede configurarse su opacidad.	
Condiciones de Ejecución: El usuario debe comprobar que se cumple la propiedad de opacidad para el componente.	
Entradas/ Pasos de Ejecución: - Seleccionar el componente. - Abrir el inspector de propiedades. - Seleccionar opción de opacidad. - Verificar que la opción de opacidad del componente funcione sobre el componente seleccionado.	
Resultado esperado: La opción de opacidad para el componente es configurable.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 17: "Caso de Prueba Aceptación #3."

Caso de Prueba Aceptación	
Número: 3	**Historia de usuario: 3**
Nombre: Configurar la dimensión de ancho componente.	

Descripción: Comprobar que se puede configurar la dimensión del componente, definido en el rango (- 2147483647; 2147483647).
Condiciones de Ejecución: El usuario debe comprobar que el objeto puede ser configurado en ancho.
Entradas/ Pasos de Ejecución: - Seleccionar el objeto. - Abrir el inspector de propiedades - Selecciona en el inspector de propiedades la opción para aumentar o disminuir el ancho del componente. - En otro caso con el clic izquierdo seleccionar uno de los puntos donde puede redimensionarse para agrandarlo o disminuirlo.
Resultado esperado: El componente se redimensiona sin perder calidad utilizando las dos opciones.
Evaluación de la prueba: Una vez realizada la primera iteración detectaron varias no conformidades, a las cuales se les da solución y se realiza otra iteración que arrojó una evaluación satisfactoria.

Tabla 18: "Caso de Prueba Aceptación #4."

Caso de Prueba Aceptación	
Número: 4	**Historia de usuario: 4**
Nombre: Configurar la dimensión de altura del componente.	

Descripción: Comprobar que se puede configurar la dimensión del componente, definido en el rango (- 2147483647; 2147483647).
Condiciones de Ejecución: El usuario debe comprobar que el objeto puede ser configurado en altura.
Entradas/ Pasos de Ejecución: - Seleccionar el objeto. - Abrir el inspector de propiedades - Selecciona en el inspector de propiedades la opción para configurar la altura del componente. - En otro caso con el clic izquierdo seleccionar uno de los puntos donde puede redimensionarse para modificar la altura del componente.
Resultado esperado: El componente se redimensiona sin perder calidad.
Evaluación de la prueba: Una vez realizada la primera iteración detectaron varias no conformidades, a las cuales se les da solución y se realiza otra iteración que arrojó una evaluación satisfactoria.

Tabla 19: "Caso de Prueba Aceptación #5."

Caso de Prueba Aceptación	
Número: 5	**Historia de usuario: 5**
Nombre: Configurar la posición (x, y) del componente.	

Descripción: Comprobar que el objeto se traslada sobre cualquier punto, definido en el rango (- 2147483647; 2147483647).
Condiciones de Ejecución: El usuario debe comprobar que el componente se puede mover sobre el despliegue.
Entradas/ Pasos de Ejecución: - Seleccionar el componente en el menú contextual. - Ubicarlo en el despliegue. - Seleccionar el componente y trasladarlo sobre cualquier punto.
Resultado esperado: El componente se traslada sobre el despliegue en cualquier punto.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 20: "Caso de Prueba Aceptación #6."

Caso de Prueba Aceptación	
Número: 6	**Historia de usuario: 6**
Nombre: Permitir la rotación del componente.	
Descripción: Comprobar que el componente rota sobre el eje z.	
Condiciones de Ejecución: El usuario debe comprobar que el componente rota sobre el eje z.	

Entradas/ Pasos de Ejecución: - Seleccionar el objeto. - Seleccionar en el menú contextual la opción rotar izquierda. - Seleccionar en el menú contextual la opción rotar derecha.
Resultado esperado: El usuario puede rotar el componente sobre cualquiera de las dos opciones.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 21: "Caso de Prueba Aceptación #7."

Caso de Prueba Aceptación	
Número: 7	**Historia de usuario: 7**
Nombre: Permitir Eliminar el componente.	
Descripción: Comprobar que el componente se elimine.	
Condiciones de Ejecución: El usuario debe comprobar que el componente sea eliminado del sistema	
Entradas/ Pasos de Ejecución: - Seleccionar el Componente. - Clic derecho encima del componente - Selecciona la opción de eliminar el componente. Otra vía:	

- Presionar la tecla Delete en el teclado y se elimina el componente.
Resultado esperado: El usuario pudo eliminar el componente sobre cualquiera de las dos opciones.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 22: "Caso de Prueba Aceptación #8."

Caso de Prueba Aceptación	
Número: 8	**Historia de usuario: 8**
Nombre: Añadir animación de blinkeo en el componente.	
Descripción: Comprobar que el componente se le añade la animación de blinkeo.	
Condiciones de Ejecución: El usuario debe comprobar que el componente	
Entradas/ Pasos de Ejecución: - Seleccionar el Componente. - Abrir El inspector de propiedades - Selecciona la opción animatable en el inspector de propiedades. - Se hace clic en el botón umbral y se añade un umbral al componente - Selecciona el nombre - Se hace clic en el botón adicionar y adicionas las condiciones (Para adicionar las condiciones se debe tener en cuenta los (Puntos, Tipo, Comparación, Valor, Condición). - Se Hace clic en el botón ok para guardar el umbral.	

Resultado esperado: El usuario pudo añadir la animación de blinkeo a los componentes.
Evaluación de la prueba: Una vez realizada la primera iteración detectaron varias no conformidades, a las cuales se les da solución y se realiza otra iteración que arrojó una evaluación satisfactoria.

Tabla 23: "Caso de Prueba Aceptación #9."

Caso de Prueba Aceptación	
Número: 9	**Historia de usuario: 9**
Nombre: Configurar color del borde del componente.	
Descripción: Comprobar que el color del borde pueda ser configurable para diferenciarlo de los demás atributos del componente.	
Condiciones de Ejecución: El usuario debe comprobar que el color del borde del componente pueda ser cambiado por otro color.	
Entradas/ Pasos de Ejecución: - Seleccionar el componente. - Abrir inspector de propiedades. - Seleccionar opción Color de borde. - Seleccionar color del panel de colores.	
Resultado esperado: El color del borde puede ser cambiado por otro cualquiera.	

Evaluación de la prueba: Una vez realizada la primera iteración detectaron varias no conformidades, a las cuales se les da solución y se realiza otra iteración que arrojó una evaluación satisfactoria.

Tabla 24: "Caso de Prueba Aceptación #10."

Caso de Prueba Aceptación	
Número: 10	**Historia de usuario: 10**
Nombre: Configurar color de fondo del componente	
Descripción: Comprobar que el color de fondo puede ser configurable para diferenciarlo de los demás atributos del objeto.	
Condiciones de Ejecución: El usuario debe comprobar que el color de fondo pueda ser cambiado por otro color**.**	
Entradas/ Pasos de Ejecución: - Seleccionar el componente. - Abrir inspector de propiedades. - Seleccionar opción Color de fondo. - Seleccionar color del panel de colores.	
Resultado esperado: El color de fondo del componente puede ser cambiado por otro cualquiera.	

Evaluación de la prueba: Una vez realizada la primera iteración detectaron varias no conformidades, a las cuales se les da solución y se realiza otra iteración que arrojó una evaluación satisfactoria.