Universidad de las Ciencias Informáticas
Facultad 5 Entornos Virtuales

Módulo de Visualización de Gráficos Vectoriales para aplicaciones SCADA

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Leonel Salazar Videaux

Raúl Pérez-Alejo Neyra

Tutor: Ing. René López Baracaldo

Ciudad de la Habana, Mayo 2007

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Raúl Pérez-Alejo Neyra

_____

Leonel Salazar Videaux

_____

René López Baracaldo

_____

Datos de Contacto

Tutor:

Ing. René López Baracaldo

Email: rene@uci.cu

Años de Graduado: 3

Años de Experiencia en el tema: 3

Dedicatoria

A Bárbara, Victoria, Diosbel y Bernardo.

Leonel.

Dedico este trabajo de diploma a toda mi familia, especialmente a mi mamá por ser la autora intelectual de este trabajo, a mi querido padre y a mi segunda mamá Carmen, a mis abuelas amadas Maria y Zenaida, a mi abuelo Marcelino, a mis tías queridas Silvia, Xiomara y Anita, a todos mis primos; Yanet y sus preciosas jimaguas Emily y Eleny, a mi amado primo Angel, a mis primas del alma Carmen y Anabel.

A mis amigos de la escuela y del alma por siempre, por su incondicional amistad.

A mis amigos del barrio, Damian, Alien y a los de Santo Suárez, a todos los quiero mucho.

A mi compañero de tesis Leonel ¡¡¡¡ya cumplimos el sueño!!!!!

Raul.

Agradecimientos

De Raul:

A mi mamá, por darme tanto cariño, ser mi musa inspiradora con tantos ejemplos de genialidad y sacrificios, por estar en el lugar preciso para un consejo oportuno.

A mi papá, por ser un ejemplo de padre y un paradigma a seguir.

A Carmen, por todo el cariño que me ha profesado durante toda su vida.

A Fung y Rene, por ayudarme a crecer humana y profesionalmente.

A todos los amigos de EMEDIA y SCADA, por todos los ratos agradables que hemos pasado juntos.

Al Comandante en Jefe por darme la oportunidad de estudiar en esta magnífica universidad.

De Leonel:

Gracias mami y abuela por su confianza y apoyo, sin ustedes no lo hubiera logrado, a Bernardo mi cariño y respeto por regalarme la familia que hoy tengo, a mi hermano "Pirri", tu también escribirás estas líneas muy pronto.

A mis amigos de siempre, Omaro, JuanK, Karel, Alberto, Reina, Irina, ..., son muchos.

A mis compañeros y amigos de SIMPRO, de SCADA y de la UCI, que tanto tiempo compartimos y disfrutamos.

A Yani, que compartió conmigo todos estos años de universidad, muchas gracias.

A "El Pillo", lo logramos.

A Fidel, que ha hecho tanto por todos los cubanos y que creó la UCI para nosotros.

Resumen

El presente documento refleja una investigación sobre las tecnologías que más se utilizan en el ámbito del software libre relacionadas con la construcción de Interfaces Hombre-Máquina en sistemas SCADA, orientándola hacia la industria venezolana de petróleo: PDVSA. El objetivo de este trabajo es obtener una aplicación de software capaz de visualizar distintos tipos de gráficos de una manera eficiente mediante una interfaz gráfica de usuario amigable, que permita la interacción con los componentes representados y que sea accesible a los usuarios en múltiples idiomas. Para cumplir esta meta se describe una solución partiendo del diseño de software y se detalla la implementación del sistema, validándola mediante la realización de pruebas de unidad al código fuente resultante.

Tabla de Contenidos

Introducción

Controlar centímetro a centímetro los procesos que se realizan en la industria ha sido una premisa para el hombre moderno. La supervisión es un tema medular para todo empresario que quiera salir adelante con productos competitivos y de calidad.

Los primeros mecanismos de control eran sencillos sistemas de telemetría que sólo proporcionaban reportes de las condiciones del campo y brindaban parámetros de las unidades remotas, solo ofrecían algunas funciones de monitoreo simple. La visión del operador que supervisaba estaba basada en contadores que se encontraban detrás de un panel de recolección de datos.

El advenimiento de los nuevos avances tecnológicos trajo soluciones eficientes al problema. Las computadoras agregaron la capacidad de programar tareas para realizar funciones más complejas y a su vez se han ido perfeccionando y adaptando a las condiciones que imponen el paso de las Tecnologías de la Información y las Comunicaciones (TIC) a nivel mundial.

Un ejemplo son los Sistemas de Control Industrial (ICS), término general que encapsula a varios tipos de sistemas de control como los SCADA, acrónimo de *Supervisory Control and Data Adquisition* (en español, Control Supervisor y Adquisición de Datos). Los primeros fueron diseñados para cumplir funciones específicas dentro de un proyecto determinado, es decir se desarrollaban para una industria en particular, las empresas que los producían se dieron cuenta que no estaban cumpliendo con varios de los principios de la programación como la adaptabilidad, escalabilidad, y la reutilización del trabajo realizado, entonces empezaron a hacer módulos generales con capacidades requeridas comúnmente y otros para aspectos más específicos, que se adaptaran a cualquier ambiente con un mínimo de cambio, fue un paso significativo aunque previsible.

Hoy día los SCADAS son parte integral de la estructura de las industrias y no son vistos como una simple herramienta operacional, sino como un recurso importante de información. Funcionan como centro de responsabilidad, al punto de poder controlar situaciones

excepcionales como desastres ecológicos en caso de industrias que trabajen con materiales contaminantes; pérdidas de vidas humanas en explosiones o escape de gases tóxicos. Proporcionan gran cantidad de datos que se entregan a sistemas o usuarios fuera del ambiente de control con los que se realizan análisis de diferentes tipos, estudios de comportamientos, detección de irregularidades y toma de decisiones.

En Cuba existe una experiencia previa en el desarrollo, implementación y despliegue de SCADA en industrias nacionales, un ejemplo es EROS, desarrollado por ingenieros de la Unión del Níquel en la provincia de Holguín en el año 1991; utilizado en más de 27 plantas de diferentes tipos: industrias relacionadas con el sector niquelífero, centrales azucareros nacionales y 3 en Brasil, incluso hasta un hotel en Cayo Coco, la mayoría instalaciones del MINBAS y el MINAZ. Es un sistema basado en tecnologías Windows que implementa muchas de las funcionalidades que exige un sistema distribuido y de control de este tipo.

En el año 2002 como parte de un plan para desestabilizar el país se produjo un sabotaje a la industria petrolera venezolana por parte de elementos contrarios al proceso revolucionario iniciado por el presidente Hugo Chávez. El atentado consistió en inhabilitar los SCADA de las distintas instalaciones aprovechando que poseían las contraseñas de acceso y control del sistema. En un estado inicial se cumplió el objetivo de la contrarrevolución, pero la acción del personal de PDVSA contrarrestó los daños hechos con acciones en contra del paro.

Esta situación sirvió como base para que el gobierno venezolano emprenda un plan de acciones para alcanzar su independencia y soberanía tecnológica de las compañías que apoyaron el sabotaje, realizar convenios con vista al desarrollo de software que sustituyen los que actualmente se utilizan, a esto se le suma la intención de migrar todas las aplicaciones del nivel gubernamental a tecnologías libre de patentes.

Dado estos factores y bajo los acuerdos suscritos por la República Bolivariana de Venezuela y la Repúplica de Cuba, se firma un contrato con la Universidad de las Ciencias Informáticas (UCI), institución que va a la vanguardia en la consolidación de la industria del software

nacional, para desarrollar un producto que sustituya los actuales SCADA e incluirse en el selecto grupo de empresas que diseñan, implementan y despliegan estos sistemas internacionalmente. Este trabajo ha sido supervisado por especialistas que fueron parte del equipo de desarrollo del SCADA EROS.

Como parte del engranaje que conforma el sistema, una de sus piezas es el modulo de ejecución de los despliegues, conocido como Interfaz Hombre-Máquina (HMI), en el que se representan mediante gráficos vectoriales y animaciones todos los procesos y datos significativos que están en el proceso; la necesidad de este modulo como parte fundamental de los requisitos funcionales ha llevado a plantear el siguiente problema:

¿Cómo desarrollar las funcionalidades del módulo de ejecución (interfaz hombre-máquina (HMI)) del sistema SCADA que se esta constuyendo con el uso de software libre?

Para darle solución al problema anterior se propone como **objeto de estudio** el ambiente de ejecución del sistema SCADA y como **campo de acción** la representación e interacción por medio de la visualización de imágenes y gráficos vectoriales de los procesos que son mostrados por el módulo HMI.

Después de este análisis se plantea como **objetivo general** la obtención de un módulo del SCADA Nacional para la visualización de las funcionalidades de este sistema utilizando tecnologías de software libre.

Para cumplir con este objetivo general se plantean las siguientes tareas:

- Realizar un estudio del estado del arte y elaborar una fundamentación teórica sobre las interfaces hombre máquina de los sistemas SCADA.
- Realizar un estudio y selección de tecnologías adecuadas para el desarrollo.
- Codificar el conjunto de conceptos definidos por el grupo de análisis y diseño.

- Diseñar las pruebas de unidad correspondientes a todos los conceptos codificados.
- Evaluar el resultado de las pruebas de unidad previamente diseñadas.

Para tener una linea inicial en nuestra investigación identificamos las siguientes preguntas científicas:

- ¿De las bibliotecas disponibles para generar interfaces gráficas (GUIs) con licencia de software libre cúal es la que más se ajusta a las necesidades del desarrollo del módulo HMI?
- ¿Qué biblioteca de gráficos vectoriales 2D puede ser utilizada para el desarrollo de la base gráfica del módulo HMI?
- ¿Qué técnica se puede utilizar para garantizar que la aplicación sea multilingue?
- ¿Qué técnica de representación de imágnes sería la más eficiente para los gráficos vectoriales y de mapa de bits?

Si se seleccionan las herramientas adecuadas con las que se le puedan dar cumplimiento a las tareas planteadas, libres de patentes comerciales y se hace un buen uso de ellas en función de darle solución al problema cientifico, entonces se obtendrá el módulo de Interfaz Hombre-Máquina (HMI) para el sistema SCADA Nacional PDVSA.

El presente documento está dividido en tres capítulos y ellos a su vez en epígrafes y sub-epígrafes temáticos que abordan temas particulares. El primer capítulo define los conceptos fundamentales sobre las tecnologías y temas que se abordaron en la concepción del sistema, justifica algunas decisiones y describe las herramientas y metodologías empleadas en la construcción del software. En el segundo capítulo se describe la solución propuesta a partir del análisis del software, se realiza una descripción detallada de las clases del diseño y sus relaciones. Un tercer y último capitulo se encarga de validar mediante pruebas realizadas al código fuente la solución implementada para el cumplimiento de los objetivos. Por último las conclusiones del trabajo, recomendaciones propuestas, bibliografía utilizada y términos importantes que nos brindan información relevante del trabajo realizado.

Capítulo 1 – Fundamentación Teórica

Introducción

Este capítulo resume los conceptos fundamentales de la teoría de los sistemas SCADA, desde las generalidades hasta las especificaciones del hardware. Además comprende la descripción de las principales tecnologías que se emplean en la construcción de estos tipos de software incluyendo la metodología, el lenguaje de modelado y las herramientas de desarrollo empleadas.

1.1 - Generalidades de un sistema SCADA

El objetivo principal de la automatización es disminuir la intervención del operador humano en la ejecución de los procesos de producción de las empresas. Con vista a cumplir esa exigente meta se han desarrollado en los últimos años algunos sistemas denominados SCADA [1].

Los sistemas SCADA son aplicaciones de software diseñadas especialmente para controlar la producción, proporcionando comunicación con los dispositivos de campo y controlando las variables y los procesos involucrados en la producción desde la pantalla de un ordenador. Además provee toda la información que se genera en el proceso productivo a diferentes usuarios: supervisores de control de calidad, mantenedores, asesores, operadores y otros.

En este tipo de sistemas interactúan ordenadores, unidades remotas, sistemas de comunicación que efectúan las tareas de supervisión, gestión de los datos y el control total de los procesos.

Un sistema SCADA está compuesto por 3 elementos fundamentales [1]:

- múltiples Unidades de Terminal Remota (conocidas como RTU)

- estación Maestra y ordenador con interfaz hombre-máquina (HMI)
- infraestructura de comunicación

La adquisición de los datos comienza en las terminales remotas, la información recopilada es formateada de manera que los operadores en el centro de control puedan supervisar el proceso utilizando la interfaz hombre-máquina, ajustar los valores de los controles en el campo y manipular las alarmas que se generen en el sistema en general, todo esto en tiempo real.

Estos sistemas adicionaron el acceso al historial de alarmas y variables, análisis de las bases de datos para generar reportes de múltiples tipos e incluyeron nuevas interfaces hombre-máquina que hacen que el sistema sea amigable.

1.1.1 – Características de un sistema SCADA

Un SCADA se comunica con los dispositivos de campo (controladores lógicos programables o PLC, autómatas, sistemas de dosificación, entre otros) para controlar el proceso desde la pantalla de un ordenador que es configurada y que puede ser modificada con facilidad. Brindan una nueva característica de automatización que hace a los SCADA diferentes de otros sistemas: la supervisión. Este aporte ha dejando atrás la capacidad de monitorear las variables y ha introducido la opción de actuar sobre el valor de las mismas en tiempo real.

En los SCADA se utilizan además HMI interactivas, que permiten al operador detectar las anomalías o alarmas en el proceso y mediante la pantalla del ordenador solucionar el problema ejecutado acciones en tiempo real, algunas de ellas sugeridas por el propio SCADA, muy distinto a otros que resuelven estas situaciones mediante el paro del proceso productivo y el reinicio del

mismo cuando se corrige el error.

Otras de las características son la adquisición y el almacenamiento de información en bases de datos de tiempo real, representación gráfica de los procesos de manera animada, creación de gráficos de tendencia a partir de los valores de las variables en el tiempo, conectividad con otros sistemas (locales y compartidos) y explotación de los datos para la gestión de la calidad, control estadístico, administración y generación de informes. Son sistemas con arquitectura abierta, capaces de crecer o adaptarse según las características cambiantes de las empresas, que se relacionan con total facilidad y de forma transparente al usuario con los equipos de campo y el resto de los ordenadores involucrados en el proceso, programas en general fáciles de instalar y configurar, con pocas exigencias de hardware y que muestren interfaces amigables a los operadores y demás interesados.

SCADA es en sí la unión de diferentes softwares que facilitan la adquisición, supervisión, control y transmisión de datos de uno o varios procesos, estos softwares se agrupan en los módulos:

- configuración, permite al operador definir el ambiente en el que trabaja del SCADA, adaptándolo al proceso particular que se quiere supervisar
- interfaz gráfica de usuario, conocida como interfaz hombre-máquina, brinda las opciones de supervisión, muestra en la pantalla de un ordenador los procesos en tiempo real mediante sinópticos definidos en un editor incorporado en el SCADA
- proceso, ejecuta acciones de control pre-programadas a partir de los datos que se adquieren en el campo
- gestión y archivo de los datos, se encarga del almacenamiento y procesado de los datos de manera ordenada, garantizando que otras aplicaciones o dispositivos tengan acceso a los registros históricos de datos

- comunicación, se ocupa de transmitir la información entre todos los componentes en la arquitectura del SCADA, incluyendo las estaciones de gestión de la administración de la empresa

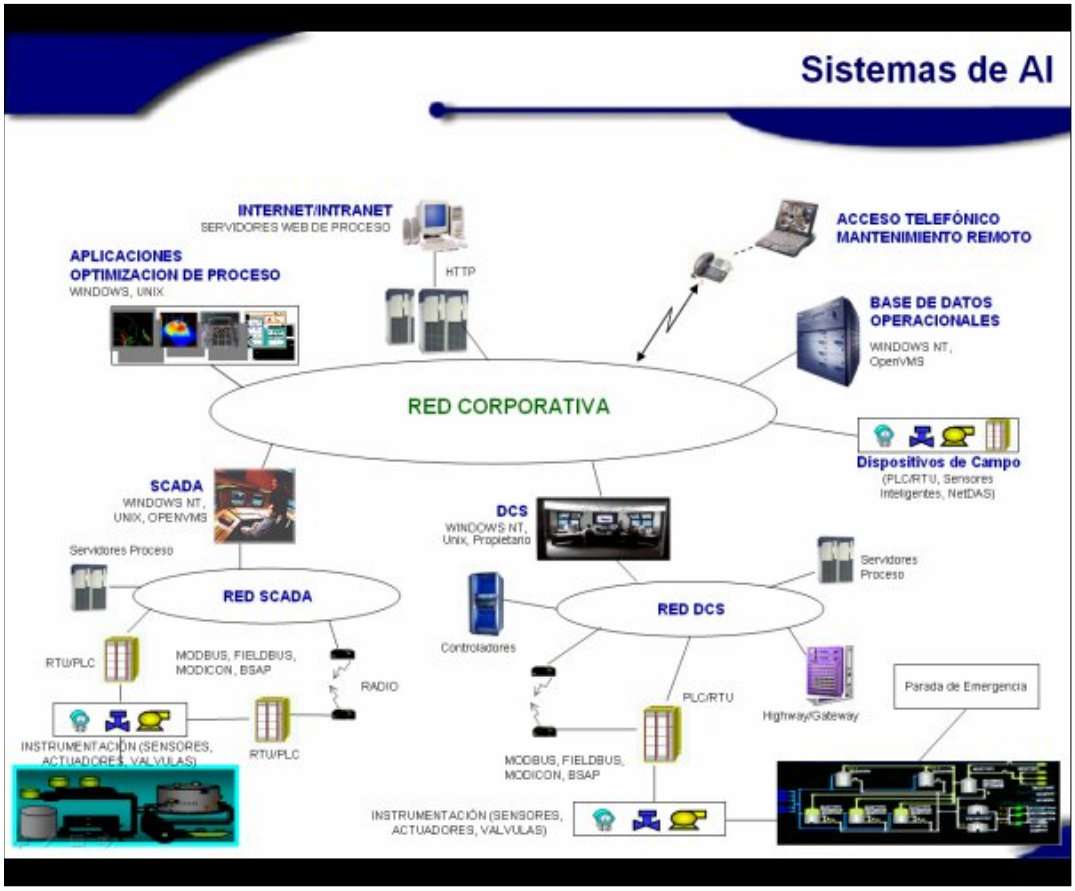

Figura 1: Relación de un sistema SCADA y el entorno.

1.1.2 – Componentes de hardware en un SCADA

Como elemento de software necesita ciertos componentes de hardware para gestionar la

información captada en el campo. Para algunas instituciones, los costos de un fallo en un dispositivo de control son extremadamente elevados, incluyendo la pérdida de la vida en algunos casos.

El hardware que se emplea en un sistema SCADA es resistente a las extremas temperaturas, vibraciones, presión, voltajes, elementos químicos y otros agentes presentes en el campo. Independientemente de estas medidas de seguridad, se suele tener hardware de respaldo y se usan varios canales de comunicación, en caso de falla en algún elemento entra en acción el mecanismo de respaldo, otro dispositivo de hardware asume la responsabilidad y el proceso continúa con costos mínimos.

El hardware involucrado en los sistemas SCADA se clasifica en cuatro grandes grupos, cada uno cumple con funciones específicas que se describen a continuación.

La Unidad Central (MTU) es el ordenador principal del sistema, generalmente es una computadora personal que contiene la HMI. La MTU actúa como intermediaria con el operador e incluye la presentación de los datos y la administración de las alarmas en tiempo real. Se encuentra en el nivel superior o nivel de gerencia en un SCADA.

Las Estaciones Remotas (RTU) están situadas en los nodos estratégicos del sistema gestionando y controlando las sub-estaciones, reciben los datos de los sensores de campo y le envían a los mismos los comandos que llegan desde la estación central. Se encuentran en el nivel intermedio o nivel de automatización.

La Red de Comunicación gestiona la información que se envía a la red de ordenadores. El tipo de

implementación es variada y depende de las necesidades y del software escogido para el sistema. Debido a que los componentes pueden estar localizados en diversas áreas, se utilizan sistemas de tipo Red de Área Amplia (Wide Area Network, WAN).

Los instrumentos de campo son todos aquellos dispositivos que permiten tanto la supervisión y el control (PLC, controladores y actuadores) así como la adquisición de los datos (sensores) en el campo. Para la selección de un sistema SCADA es necesario tener en cuenta aspectos como:

- número de variables del proceso a automatizar
- si el proceso está distribuido geográficamente
- si se requiere o no la capacidad de operar en tiempo real

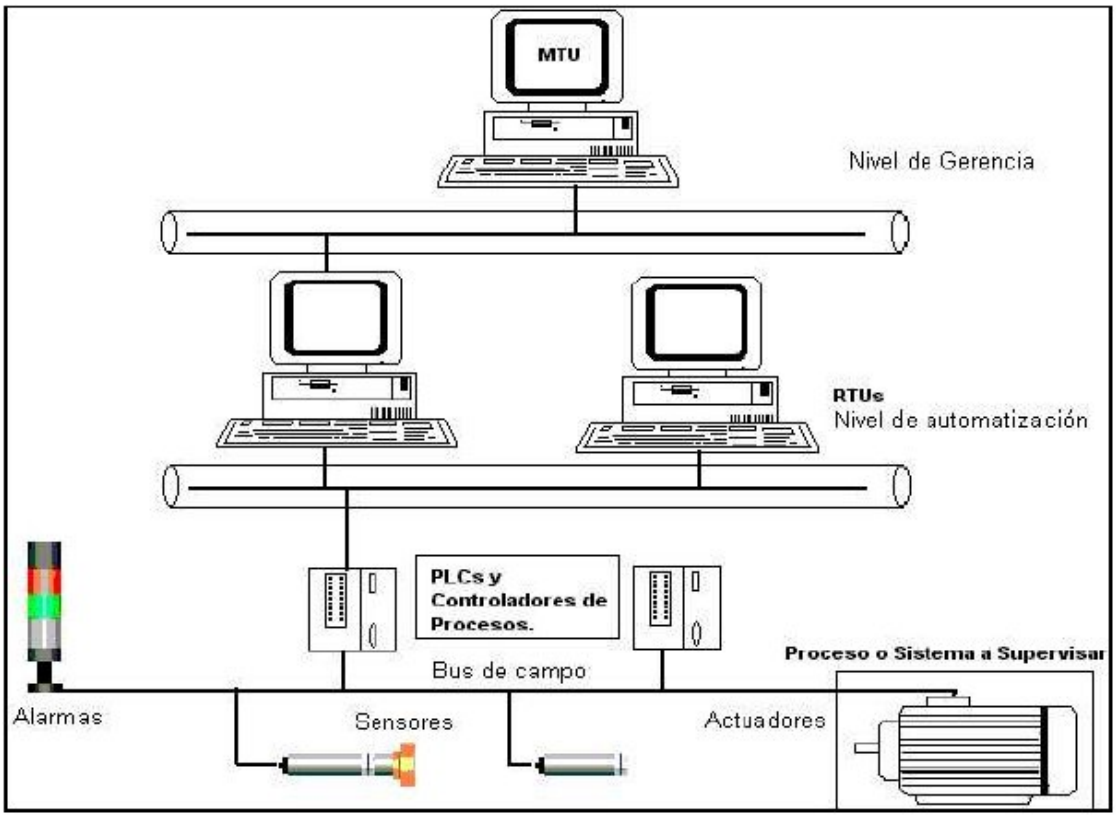

Figura 2: Organización del SCADA en niveles.

1.1.3 – Ejemplos de sistemas SCADA

Hoy en día existen varios sistemas SCADA, algunos muy reconocidos internacionalmente por el prestigio de sus proveedores. A continuación se mencionan algunos de los SCADA que se comercializan en el mundo:

- HMI/SCADA Paragon, de Nematron
- HYBREX(Hybrex Expert System), de SIEMENS
- LabView, de National Instruments

Los SCADA anteriormente mencionado son totalmente propietarios. También se han desarrollado algunos de estos sistemas utilizando tecnologías libres, teniendo en cuenta el incremento del avance que cada día protagonizan los software que apuestan por el código abierto. A continuación se muestran algunos proyectos que han tenido notables resultados:

- FreeSCADA, de Raditex AB [2]
- Visual [3]
- Qscada [4]

1.2 – El módulo de Interfaz Hombre-Máquina

El término Interfaz Hombre-Máquina es usado frecuentemente en el contexto de los Sistemas de Computación y los Sistemas Electrónicos, constituye una capa intermedia que los independiza y permite la intercomunicación entre ambas partes.

Existen en la actualidad diferentes tipos de HMI, se pueden identificar claramente dos de ellas:

Interfaces Gráficas de Usuario (GUI) y las Interfaces de Usuario basadas en la Web. Es interesante adicionar que hay variados tipos de estas interfaces que actualmente se utilizan en menor escala, por ejemplo: interfaces basadas en líneas de comandos, táctiles, basadas en gestos, multi-pantallas e interfaces basadas en texto.

Los sistemas SCADA de la actualidad poseen varias HMI, estas interfaces emplean técnicas de visualización de elementos gráficos, anteriormente de tipo rasterizadas y ahora de tipo vectorial. Las HMI pueden ser GUIs o basadas en Web, dependiendo del interesado en supervisar el proceso en cuestión.

El módulo de HMI en el SCADA se encarga de representar, en un ordenador, los procesos que ocurren en el campo en tiempo real, muestra los componentes implicados, los sensores, las estaciones remotas, y el sistema de comunicación dándole al operador total control. Éste módulo es el que permite al operador estar en contacto directo con el sistema y realizar la supervisión y el control del proceso en general.

Está compuesto por dos partes fundamentales, el ambiente de configuración o editor y el ambiente de ejecución, éstas pueden ser aplicaciones separadas o estar incluidas en una sola.

El primero de estos permite configurar varios procesos o partes de ellos, aquí se definen y gestionan las variables, los drivers, los comandos, las alarmas y variadas opciones adicionales. Este ambiente funciona como una aplicación de diseño tradicional, con la peculiaridad que los sinópticos se confeccionan a partir de objetos y primitivas básicas predefinidas, que se pueden agrupar, combinar, transformar, importar y exportar entre otras.

El ambiente de ejecución se puede comparar con un reproductor multimedia, se encarga de visualizar las animaciones y los objetos definidos en el editor, muestra lo que esta ocurriendo en el campo en tiempo real, es el que envía los comandos a las estaciones remotas, quién recibe los valores de las variables, es el que interactúa con la mayoría de los operadores pues se emplea para supervisar el proceso de manera directa.

Al ser el módulo que se encarga de brindar el control total sobre el proceso de producción, la interfaz de usuario brinda un conjunto de funcionalidades primarias, entre ellas la generación de reportes, impresión, análisis de variables, graficación de la tendencia de indicadores, configuración de los drivers para la comunicación y acceso a las alarmas.

1.3 – Tecnologías libres más utilizadas

En el mundo del software libre se emplean varias tecnologías que permiten a los desarrolladores utilizarlas dependiendo de las necesidades y de las características de cada sistema que se construya. Los sistemas libres están adentrándose con creces en el ambiente de las interfaces gráficas de usuario, dejando atrás el mito del sistema operativo en modo texto, para ello emplean diferentes paquetes de desarrollo y además variadas tecnologías para la representación de los elementos gráficos, buscando elevar la eficiencia y calidad en el renderizado de las imágenes.

Este epígrafe resalta las características principales de las tecnologías libres empleadas para construir software de código abierto relacionadas con los gráficos vectoriales.

1.3.1 – GTK+

GTK+ [5] son las siglas de GIMP Toolkit, es una biblioteca que permite crear interfaces gráficas de

usuario, se distribuye bajo la licencia pública general (GPL), lo que hace de GTK+ un producto completamente libre.

GTK+ es multi-plataforma, se ha extendido hasta Microsoft Windows y muchos derivados de Unix, como Linux y Mac OS. Está escrita en C y tiene extensiones en varios lenguajes como C++, Python, Perl y muchos más.

Se empleó inicialmente en el proyecto Gnu Image Manipulation Program Tool Kit, por eso su nombre: GTK+, se ha extendido rápidamente por su estabilidad y una de las implementaciones que más se ha usado es la de C++, llamada Gtkmm[6].

Gtkmm es la implementación oficial de GTK+ escrita en C++, le adiciona a GTK+ las potencialidades del paradigma orientado a objetos, la herencia para crear nuevos componentes, polimorfismo, manejo de memoria para construir y destruir objetos, elimina el uso de las macros de C y muchas otras mejoras.

GTK+ depende de otras bibliotecas que hacen de GTK+ un éxito total:

- Glib, una biblioteca de propósito general, no destinada a interfaces gráficas en sí, provee tipos de datos, macros y utilidades de conversión, tratamiento de cadenas y abstracciones muy útiles
- Pango, se encarga de la manipulación de los textos internacionalizados, provee wigdets que se encargan de la representación de los textos
- Atk, es el paquete de accesibilidad, provee un conjunto de interfaces que permiten a las interfaces de usuario interactuar con las tecnologías de accesibilidad

- Gdk, es la capa de abstracción que permite a GTK+ ser portable a múltiples plataformas
- GTK+, ella en sí contiene las definiciones de todos los widgets

GTK+ tiene soporte para bases de datos, el proyecto Gnome-DB ofrece una arquitectura basada en CORBA que permite acceso totalmente transparente a distintas fuentes de datos, incluyendo datos que se encuentren en servidores LDAP o en ficheros XML, entre otros. GTK+ provee también mecanismos para comunicar aplicaciones de red mediante los protocolos TCP, UDP y para el trabajo con la tecnología XML.

1.3.2 – Qt

Qt es un framework escrito en C++ que permite crear aplicaciones con interfaces gráficas. Qt es totalmente orientado a objetos, fácil de usar, extensible y multi-plataforma (soportada en Windows, Unix y derivados) [7].

Qt es un producto creado por la compañía Trolltech, esta comercializa una versión de Qt no libre para los sistemas Windows, además distribuye una versión completamente libre y gratuita para los sistemas Unix, Linux y derivados, que se distribuye bajo licencia GPL y QPL [8], que está aprobada por la Fundación de Software Libre (FSF).

Qt provee mecanismos para la visualización de imágenes vectoriales y raster, utilizando widgets para hacer el render y el procesamiento de las operaciones principales: zoom, escala, rotación, traslación y pan. Provee también mecanismos para la edición de las imágenes y los objetos gráficos en general, entre ellos cambio de brocha, pincel, estilo de los textos, terminaciones de brocha, estilos de línea entre otros.

Este *toolkit* incluye un poderoso módulo que se encarga del manejo de archivos en formato XML, proporciona un mecanismo de análisis de la estructura de un documento XML (parser) usando SAX2 y el modelo de objetos (DOM).

El framework de Qt, al igual que ha hecho GTK+, se ha extendido a otros lenguajes además de C++, existen implementaciones como PyQt, Qt Jambi, PerlQt, QtRyby para Python, Java, Perl y Ruby respectivamente.

Qt incluye módulos para la representación gráfica 3D utilizando OpenGl, otros para integrarse con otras aplicaciones utilizando los protocolos de red TCP, UDP y HTTP, para desarrollar aplicaciones relacionadas con bases de datos con soporte para MySQl, PostgreSQL, MSSQL, Oracle, SQLite e Interbase.

1.3.3 – Cairo

Cairo [9] es una biblioteca de gráficos 2D completamente libre, con soporte para múltiples dispositivos de salida, entre ellos X Window System, Win32, PostScript, PDF, archivos SVG y otros en estado de prueba como OpenGl, Quartz y XCB.

Cairo está diseñada para producir salidas consistentes en todos los dispositivos mientras emplea al máximo la aceleración del hardware de video si está disponible. La *Application Programming Interface* (API) de Cairo ofrece operaciones similares a las de dibujado empleadas en las tecnologías PostScript y PDF. Las operaciones en Cairo incluyen pinceladas, rellenos, curvas

Bézier, variadas transformaciones y representación de textos, todas las operaciones de dibujo se pueden lograr mediante las transformaciones básicas (rotar, escalar, mover).

Cairo está escrita en el lenguaje C, pero posee implementaciones para Lisp, Java, Mono, Perl, PHP, Pyton, Ruby y otros.

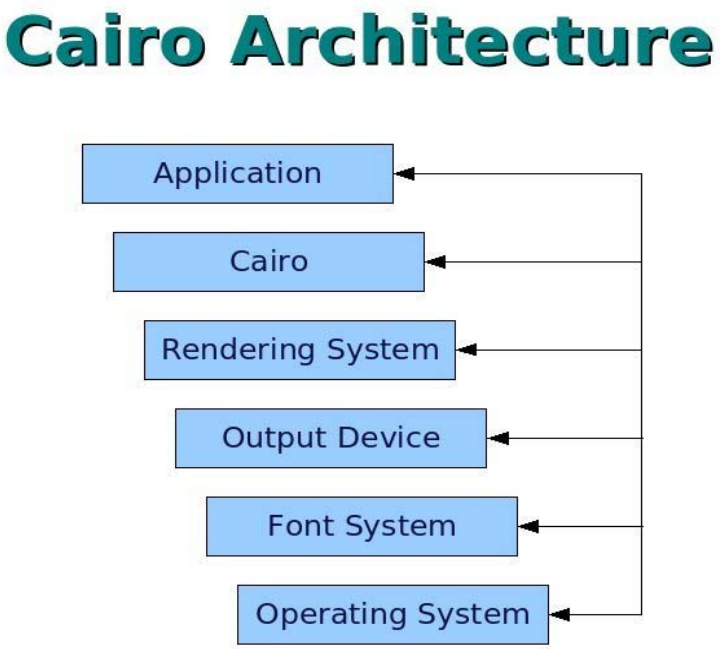

Figura 3: Arquitectura de Cairo.

1.3.4 – SVG

SVG son las siglas de Scalable Vector Graphics. Esta tecnología fue creada por el World Wide Web Consortium (W3C) para representar imágenes vectoriales en la web. Acerca de los gráficos en formato SVG Tim Berners-Lee, creador de la World Wide Web, dijo: "*(...) graphics which can*

be rendered optimally on all sizes of device" [10], o sea, "gráficos que pueden ser renderizados de forma óptima en todas las dimensiones del dispositivo".

SVG es una plataforma para gráficos en dos dimensiones (2D), está formado por dos partes fundamentales: un formato de archivo basado en la tecnología XML y una interfaz de programación para aplicaciones gráficas. Entre las principales ventajas tenemos que puede incluir textos e imágenes raster, incluye además un conjunto de formas básicas, diversos estilos para el dibujado, integración con varios lenguajes script y soporte comprensible para animaciones.

Los gráficos en formato SVG son usados en diferentes áreas incluyendo imágenes para la web, animaciones, interfaces de usuario, aplicaciones para móviles y diseños de alta calidad.

El formato SVG surgió en la W3C con el apoyo de importantes compañías líderes en el área de los gráficos, entre ellas Adobe, Apple, Canon, Corel, Kodak y Macromedia. Incluye tres tipos de objetos gráficos: formas gráficas vectoriales, imágenes en otros formatos y textos. Los objetos en un SVG pueden agruparse, transformarse y adicionarse a otros archivos gracias al estándar XML, pues los archivos SVG se escriben en texto plano.

Es importante señalar algunos conceptos importantes sobre SVG [11]:

- *Gráficos*: SVG tiene como base a los objetos completos en lugar de puntos individuales, provee un conjunto de formas básicas y una herramienta para crear trazos que se usan para realizar increíbles dibujos
- *Símbolos*: se pueden utilizar o crear sin necesidad de tenerlos almacenados en un registro central, son elaborados por comunidades de usuarios, se pueden incluir en nuestros

diseños y mantendrán su aspecto original, sin importar si son transformados

- *Efectos raster*: muchos efectos como sombras, desenfoque e iluminación se emplean en las imágenes raster, SVG además de éstos, incluye elementos de filtros, que se combinan y envían al cliente en el momento de representar la imagen, dándole aún mayor fortaleza a los gráficos vectoriales, pues no se pierde ningún efecto al redimensionar o mostrar las imágenes en diferentes resoluciones
- *Animación*: puede lograrse por medio de los lenguajes script, es complejo desarrollar un script y darle mantenimiento, por esta razón las comunidades de SVG han desarrollado scripts que crean los efectos más comunes fundamentalmente para la web

¿Por qué escoger SVG? [12]. En primer lugar porque incluye variados aspectos que lo hacen superior a los mapas de bits: el formato SVG es mucho más pequeño, es escalable, todas las formas y los textos que incluye son editables mediante curvas Bézier y soportan las fuentes más comunes; además el texto puede ser seleccionado, editado e indexado por los buscadores; se puede generar dinámicamente en un servidor como respuesta a una petición Javascript, Perl Java o XML, por solo mencionar algunas.

1.4 – Otras tecnologías libres empleadas

En el desarrollo de software usando tecnologías libres es muy común tener la posibilidad de presentar los softwares en varios idiomas, esto es posible mediante un concepto que ha tenido mucho éxito en las aplicaciones libres: la internacionalización.

Otro aspecto importante a tener en cuenta en este ámbito es el análisis de la información mediante gráficos de tendencia que es muy útil cuando se requiere llevar métricas relacionadas con la evolución de alguna variable o proceso en el tiempo.

1.4.1 – Internacionalización

La internacionalización es la práctica de diseñar y escribir programas que pueden configurarse fácilmente para interactuar con el usuario en más de un idioma. Esta funcionalidad se ha asumido con mucho ímpetu en el contexto del software libre, el gran beneficio que brinda a los usuarios finales es poder utilizar determinado software en varios idiomas dependiendo de la región o del país en que se ejecute.

Usualmente los programas se escriben y se documentan en idioma inglés, además, en tiempo de ejecución toda la interacción con los usuarios se realiza en este idioma. Escribir software en un idioma muy utilizado es una ventaja para los desarrolladores, mantenedores y documentadores de software, pero constituye una barrera para una parte importante de los usuarios, pues muchos de ellos no están familiarizados con idiomas foráneos y desean emplear software en el idioma de su lengua materna.

Existen varias bibliotecas que brindan soporte para la internacionalización, por solo mencionar dos de ellas tenemos gettext [13] e i18n [14]. Ambas están muy difundidas y ofrecen disímiles opciones para lograr la internacionalización.

La biblioteca gettext constituye un paso importante en materia de traducción de aplicaciones de software en el proyecto GNU. Este paquete ofrece a los desarrolladores, traductores e incluso a los usuarios un conjunto de herramientas que permiten producir mensajes en múltiples idiomas. Esas herramientas incluyen:

- convenciones acerca de ¿cómo escribir los programas para soportar los catálogos?
- bibliotecas para soportar la generación de mensajes traducidos en tiempo de ejecución
- algunos programas para traducir cadenas de caracteres en varios idiomas

GNU gettext esta diseñado para minimizar el impacto de la internacionalización en el código fuente de los programas. Para obtener software multi-lenguaje se incluye un grupo de aspectos que no se pueden pasar por alto dependiendo del país o región en que se encuentre el usuario final, el lenguaje nativo, el formato de la fecha y hora, las convenciones numéricas y financieras son solo algunos de ellos.

La siguiente imagen muestra un resumen de la relación entre los archivos manipulados por gettext y las herramienas que interactúan con ellos.

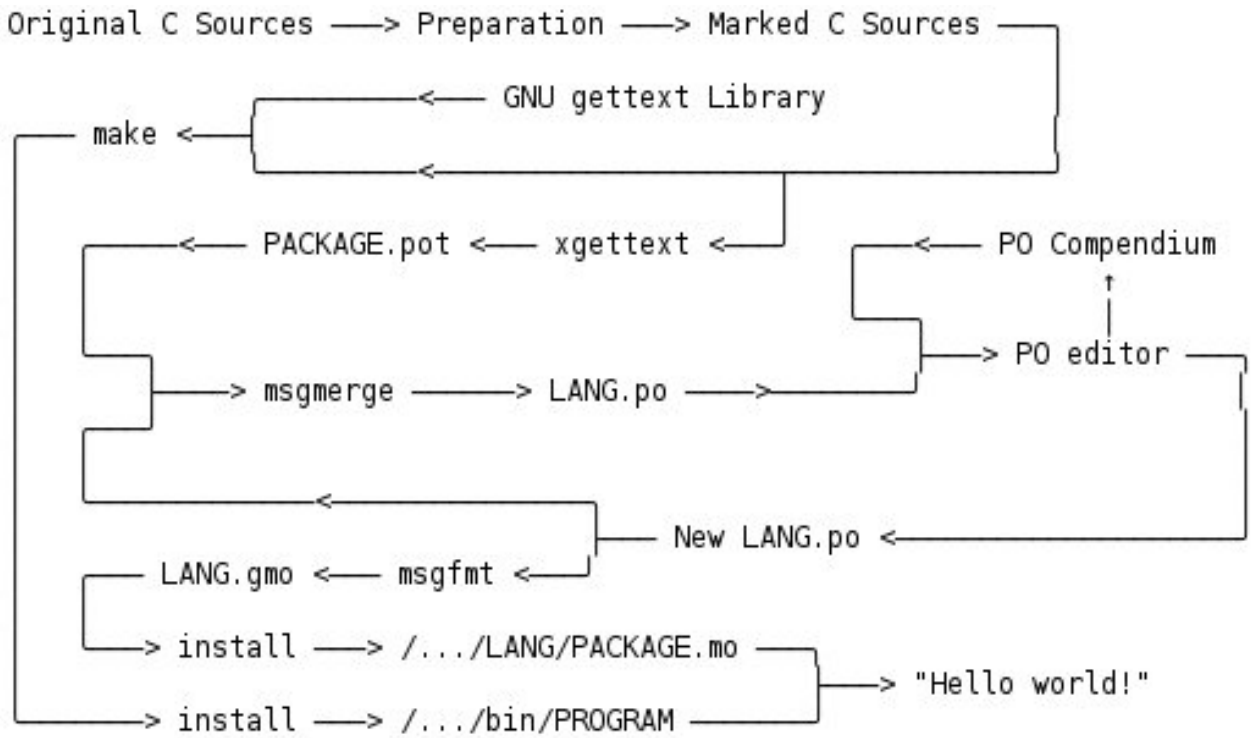

Figura 4: Relación entre gettext y los archivos a internacionalizar.

1.4.2 – Gráficos de tendencia

Los gráficos de tendencia son muy útiles en el momento de analizar fallas en los sistemas. Sirven para representar de manera gráfica las funcionalidades de un registro o “log”, emplean una base de datos que conserva todos los valores que han obtenido un grupo elevado de variables en el tiempo. La base de datos se actualiza con una frecuencia definida por el tipo de proceso que se supervisa, puede ir desde milisegundos hasta días, y los registros se pueden almacenar por períodos largos, incluyendo espacio de varios años.

La biblioteca PLplot [17] es utiliza para representar gráficos de tendencia, es un pequeña, portable, de uso libre y rica en los tipos de gráficos que soporta.

Esta biblioteca brinda un grupo de funciones escritas en C que son muy útiles para hacer ploteos científicos y representar comportamientos de valores de variables en funciones definidas por el usuario. Brinda soporte para varios lenguajes de programación como C, C++, Fortran, Java y lenguajes interpretados como Octave, Python, Perl y Tcl/Tk.

Se puede utilizar para realizar distintos tipos de gráficos en dependencia de la necesidad del programados; entre sus funcionalidades su encuentra la capacidad de realizar gráficos en 2D y 3D, gráficos de barra y de pastel, escalas de colores para las representaciones dependiendo de criterios especificados, definición de escalas de valores y notación científica automática, personalización de las lineas (color, ancho, tipo, marcas), notación científica, soporte para mas de dos mil caracteres incluyendo el alfabeto Griego, símbolos matemáticos y musicales entre otros.

Una de sus principales potencialidades es la variedad de formatos de salida que brinda (output devices), lo que la hace adaptable a cualquier ambiente: buffer en memoria, formato metafile (meta archivo), X Window System, JPG, PNG, formatos PostScript, formatos HP y la biblioteca es soportada en las plataformas: Windows, Unix, Linux y Mac Os X.

1.5 – Metodologías de desarrollo de software

Esta sección describe los conceptos fundamentales que describen el desarrollo de software, incluyendo la metodología Rational Unified Process (RUP), el Lenguaje Unificado de Modelado (UML), el lenguaje de programación C++ y el Embiente de Desarrollo Integrado Eclipse.

El Proceso Unificado es un marco de trabajo genérico que puede especializarse para gran variedad de sistemas [18].

Las principales características de RUP son:

- centrado en la arquitectura
- dirigido por casos de uso
- iterativo e incremental

RUP define cuatro fases para el desarrollo del software: inicio, elaboración, construcción y transición. Además define un grupo de flujos de trabajo básicos: Modelamiento del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba y Despliegue; incluye flujos de trabajo de apoyo como Gestión del Cambio y la Configuración, Gestión del Proyecto y Ambiente.

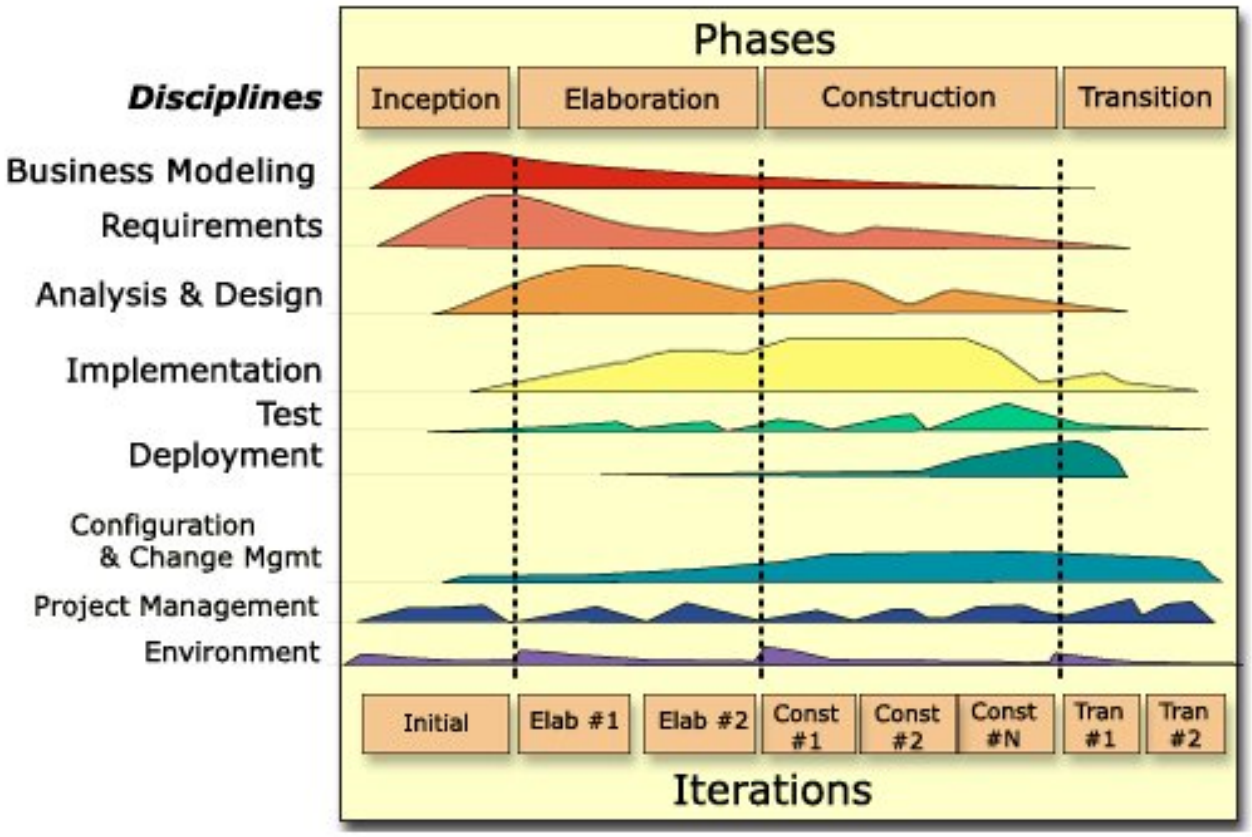

Figura 5: Fases y Flujos de trabajo de RUP.

UML [19] (Unified Modeling Language) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software.

Actualmente UML ofrece un estándar para describir un “plano” del sistema incluyendo aspectos conceptuales como procesos del negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software utilizados.

Es importante destacar que UML es un lenguaje de modelado, no un método o un proceso, se emplea para definir, detallar y documentar los artefactos de un sistema de software. Actualmente se ha publicado la versión 2.0, en esta última se adicionaron diversas novedades que resuelven carencias desde el punto de vista práctico fundamentalmente. Entre los diagramas que propone UML para modelar un sistema encontramos:

- diagramas de estructura(clases, componentes, objetos, despliegue. paquetes)
- diagramas de comportamiento(actividades, casos de uso, estado)
- diagramas de interacción(secuencia, comunicación)

1.6 – Herramientas de desarrollo empleadas

Para la implementación de estos tipos de sistema se emplean lenguajes robustos, de propósito general, que permitan la programación orientada a objetos, segura, multi-hilo, conexión a bases de datos, trabajo con imágenes, manejo de archivos entre otras funcionalidades. Además minimizar el tiempo de desarrollo es necesario utilizar algunas herramientas para el trabajo en equipo, la gestión de tiempo, el control de versiones y el desarrollo de las aplicaciones mediante entornos de desarrollo integrado.

Existen diversos candidatos que cumplen con estos y otros requerimientos, se han seleccionado un grupo de ellos (C++, Eclipse, Subversion) siguiendo criterios imprescindibles como: nivel de desarrollo y estabilidad alcanzados, documentación existente, flexibilidad y personalización, y como elemento escencial que sean aplicaciones desarrolladas bajo los terminos de las licencias que rigen el mundo del sofware libre en general.

El lenguaje de programación C++ se creó en la década de 1980 por Bjarne Stroustrup como

extensión del lenguaje C. Entre las principales características de C++ se pueden mencionar:

- soporte para la programación orientada a objetos
- programación genérica o uso de plantillas (templates)
- posibilidad de redefinir operadores
- identificación de tipos en tiempo de ejecución (RTTI)
- trabajo a alto y bajo nivel
- es un lenguaje multiplataforma

Un entorno de desarrollo integrado (IDE) muy utilizado es Eclipse, un proyecto de software libre que ha ido creciendo gracias a la fuerte comunidad que lo desarrolla. Eclipse es un IDE multi-plataforma desarrollado por IBM, en la actualidad lo mantiene la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos, capacidades y servicios complementarios.

Inicialmente Eclipse se desarrolló para los programadores que usaban el lenguaje Java, presentaba varios componentes como: la Plataforma Principal (inicio y ejecución de plug-ins), es Standar Widget Toolkit (portables widgets) y el Workbench (vistas, editores, perspectivas y asistentes). Actualmente el IDE emplea módulos (plug-ins) para adicionar funcionalidades según las necesite el desarrollador, a diferencia de otros entornos monolíticos que las tienen todas incluidas, sean necesarias o no. Gracias a estos plug-ins se ha extendido el soporte de Eclipse hasta lenguajes como C/C++, Phyton y PHP entro otros, además permite utilizar lenguajes de procesado de texto como LaTeX, aplicaciones de red como Telnet, Sistemas de Gestión de Bases de Datos (DBMS). Para la gestión de la configuración y el control de versiones tiene soporte para CVS y Subversion, incluye plug-ins para realizar pruebas de unidad (Junit, CxxTest).

Actualmente el Proyecto Eclipse exibe su versión 3.2, y entre las características fundamentales que presenta encontramos:

- editor de textos
- resaltado de sintaxis para varios lenguajes de programación
- compilación en tiempo real
- soporte para pruebas unitarias
- integración con sistemas de control de versiones

Por último el Proyecto Eclipse esta compuesto por varios sub-proyectos muy bien definidos y aceptados por su comunidad de usuarios, por solo citar algunos entontramos:

- Plataforma de Herramientas para Pruebas y Desempeño
- Plataforma de Herramientas Web
- Edición Visual
- UML2
- Plataforma de Herramientas de Datos
- Plataforma de Desarrollo de Software para Dispositivos

Conclusiones

Como resultado del análisis de las principales tecnologías que empleamos en la construcción de la solución se realizó una toma de decisiones con relación un grupo de aspectos de significativo peso para el sistema SCADA Nacional PDVSA.

Con relación a la biblioteca gráfica a emplear se optó por GTK+ como principal candidata, teniendo en cuenta la experiencia personal del equipo de desarrollo en primer plano, como elementos adicionales se presentaron los resultados de los proyectos internacionales con mayor relevancia en los ambientes de software libre, entre ellos el proyecto GNOME; otros detalles importantes resultan los términos de licencia y distribución de las bibliotecas, teniendo GTK+ una licencia totalmente libre bajo los términos de la GPL y soportada por la fundación GTK+, por su parte Qt se distribuye con más de una licencia que impone condiciones propietarias en dependencia de la plataforma en la que se emplee.

Otro debate sostenido giró entorno a la biblioteca que se utilizaría para graficar la tendencia de los indicadores, inicialmente se optó por PLPlot, en la práctica se demostró que necesitábamos obtener algunas funcionalidades que PLPlot no había cubierto plenamente, por lo que decidimos implementar nuestra propia biblioteca de gráficos de tendencia utilizando la implementación de CairoCanvas que proponemos en la solución.

Capítulo 2 – Descripción de la Solución Propuesta

Introducción

Este capítulo refleja la solución que se propone para resolver el problema científico identificado. Se hace una descripción de los módulos y componentes que se utilizan en la construcción del software. Se incluyen algunos aspectos del diseño que sirven de punto de partida para la implementación del sistema SCADA y se describe con detalles las principales clases del diseño que se utilizaron en la implementación de la interfaz hombre máquina del SCADA Nacional.

2.1 – Consideraciones generales acerca del Diseño

El diseño del sistema se basó en una arquitectura de tipo Presentacion-Abstracción-Contol (PAC) que deriva del patrón Modelo-Vista-Controlador (MVC). Es un modelo recursivo que implementa una jerarquía de PAC. Se adapta más a sistemas compuestos por sub-sistemas que necesiten una manera particular de interacción para cada caso, en sentido general favorece la modularidad del diseño. En este patrón cada objeto está solamante relacionado con su propia abstracción, la que es accesible por otros a traves de la clase controladora. El PAC permite realizar cambios con total independencia mediante la introducción de los componentes entre la capa de presentación, la de abstracción y la de control.

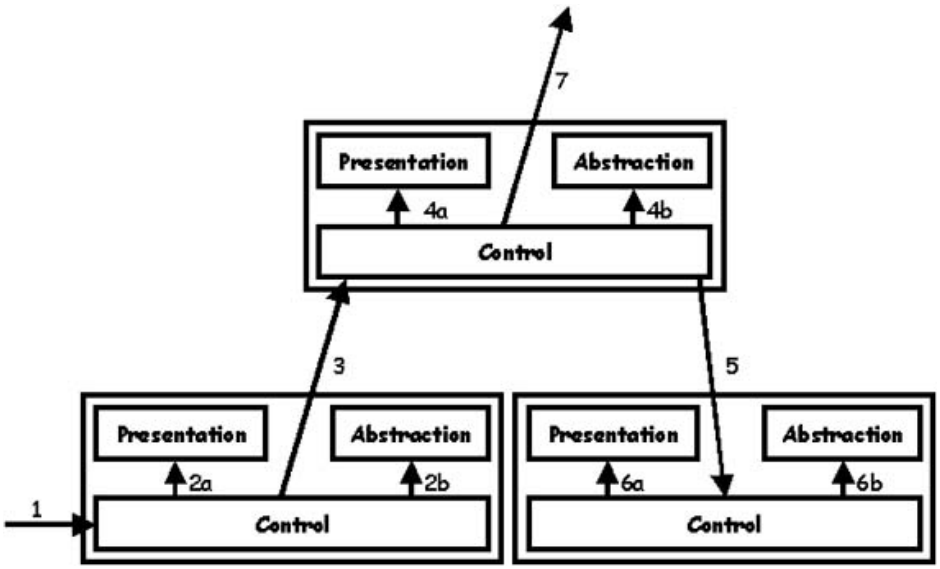

Figura 6: Flujo de operaciones en el patrón de Arquitectura PAC.

De manera general las partes del PAC tienen las siguientes responsabilidades:

- *Presentación*: se encarga de visualizar la información y tenerla lista para la interacción, se trata generalmente de una interfaz gráfica de usuario (GUI)
- *Modelo*: representa el dominio específico de la información con que se opera, presenta la estructura de la aplicación y no juega ningún papel activo en los cambios que se le hagan al comportamiento de un objeto
- *Controlador*: procesa y responde a los eventos que producen generalmente las acciones del usuario o procesos externos en sentido general y pasa éstas a su padre en la gerarquia PAC

PAC es altamente empleado en el desarrollo de software, el flujo de operaciones que representa generalmente se comporta como sigue:

1. el usuario interactúa con la GUI de alguna manera (por ejemplo: presionando un botón del ratón)
2. el controlador maneja el evento de entrada proveniente de la vista y se lo envia a su padre en la jerarquia
3. el padre actualiza todos los hijos, los que a su vez lo hacen a su presentación
4. luego de que todos ellos sean actualizados, se actualiza el padre
5. termina cuando todos los elementos han sido actualizados

2.2 – Descripción de los principales subsistemas

En este epígrafe se describen los principales subsistemas utilizados en la implementación del sistema partiendo de un empaquetamiento de las clases del diseño. De forma general se

describe cada subsistema, luego se especifica uno o varios diagramas de clases para mostrar la interacción entre las clases y por último una descripción detallada de cada clase, sus atributos y métodos.

2.2.1 - Subsistema Base

En este subsistema se encuentran las clases sobre las cuales se soportara parte del sistema, estas definen las principales interfaces que se implementarán para la visualización de los despliegues, asignación a las ventanas, fabricación y renderizado de los objetos y el control de la aplicación en general.

Como principales elementos de este subsistema se encuentran las clases Factory, Viewer, Application, Window y Screen, encargadas de crear los objetos, administrar la entrada/salida de los datos, controlar la aplicación, manejar el concepto de ventana y el de pantalla o despliegue respectivamente.

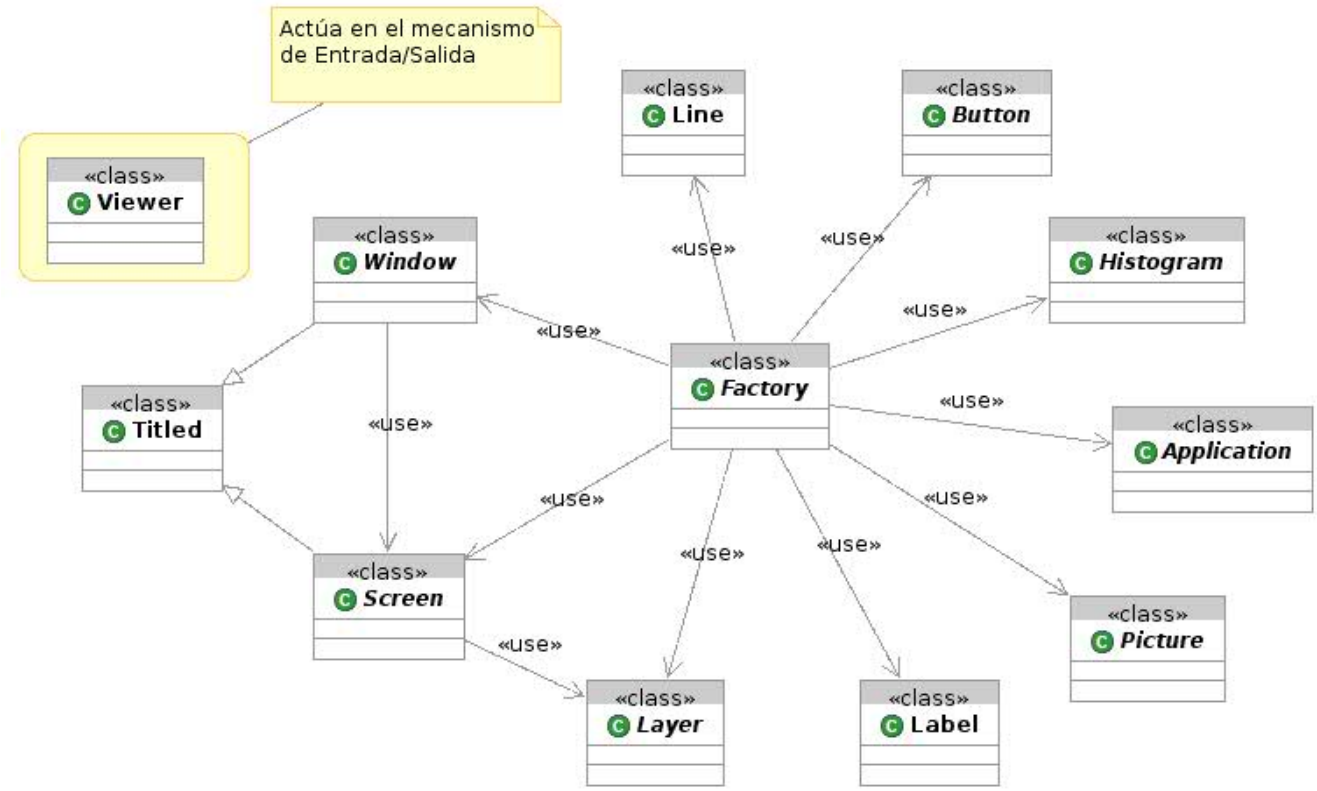

Figura 7: Diagrama de clases. Subsistema Base.

Nombre: Application	
Descripción: Define el concepto básico de una aplicación.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	void run( ) = 0
Nombre:	void setTimeOut(Viewer& viewer, TimeOutHandler handler, TimeOutInterval interval) = 0

Nombre: Factory	
Descripción: Es la fábrica abstracta de todos los objetos que se construirán en el módulo.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	Base::Window* createWindow(Identifier id, Name name) = 0
Nombre:	Base::Screen* createScreen(Identifier id, Name name) = 0
Nombre:	Graphics::Button* createButton(Identifier id, Name name) = 0
Nombre:	Graphics::Line* createLine(Identifier id, Name name) = 0
Nombre:	Graphics::Picture* createPicture(Identifier id, Name name) = 0
Nombre:	Graphics::Layer* createLayer(Identifier id, Name name) = 0
Nombre:	Graphics::Histogram* createHistogram (Identifier id, Name name)=0
Nombre:	Base::Application * createApplication() = 0

Nombre: Titled	
Descrpción: Clase base para todos los objetos que tienen un título. Hereda de la clase Named.	
Tipo de clase: Contoladora	
Atributo	**Tipo**
title	Title
Para cada responsabilidad:	
Nombre:	Title getTitle( )
Descripción:	Retorna el valor del atributo title.
Nombre:	void setTitle(Title title)
Descripción:	Asigna un nuevo valor al atributo title.

Nombre: Screen	
Descripción: Representa un mímico o despliegue de un proceso, tiene asociado varias capas que contienen los objetos gráficos. Hereda de la clase Titled.	
Tipo de clase: Controladora	
Atributo	**Tipo**
description	String
Para cada responsabilidad:	
Nombre:	String getDescription( )
Descripción:	Retorna el valor del atributo description.
Nombre:	void setDescription(String description)
Descripción:	Asigna un nuevo valor al atributo description
Nombre:	LayerCollection& getLayerCollection( ) = 0
Nombre:	void addLayer(Layer* layer) = 0

Nombre: Viewer	
Tipo de clase: Controladora	
Descripción: Es la clase principal que contiene las despliegues y las ventanas donde se mostraran, gestiona la asignación de ellos y el manejo de los datos de entrada y salida a travez del IOManager	
Atributo	**Tipo**
ioManager	IOManager*
screenCollection	ScreenCollection
windowCollection	WindowCollection
Para cada responsabilidad:	
Nombre:	IOManeger& getIOManager
Descripción:	Retorna el valor del atributo ioManeger.
Nombre:	void setIOManager(IOManager* ioManager)
Descripción:	Asigna un nuevo valor al atributo ioManager.

Nombre:	ScreenCollection& getScreenCollection( )
Descripción:	Retorna el valor del atributo screenCollection.
Nombre:	WindowsCollection& getWindowCollection( )
Descripción:	Retorna el valor del atributo windowCollection.
Nombre:	void assingScreenToWindow(Identifier screenID, Identifier windowID)
Descripción:	Se realiza la asignación de los despliegues a las ventanas donde se mostraran

Nombre: Window	
Description: Es el encargado de renderear los Screen que tiene el visualizador. Hereda de las clases Titled, Positionable y PosicionableImpl.	
Tipo de clase: Controladora	
Atributo	**Tipo**
status	WindowStatus
Para cada responsabilidad:	
Nombre:	WindowStatus getStatus( )
Descripción:	Retorna el valor del atributo privado status.
Nombre:	void setStatus(WindowStatus windowStatus)
Descripción:	Asigna un nuevo valor al atributo status.
Nombre:	Point* getTopLeft()
Descripción:	Retorna el valor del atributo topLeft heredado de la clase PositionableImpl.
Nombre:	void setTopLeft (Point* topLeft)
Descripción:	Asigna un nuevo valor para el atributo topLeft
Nombre:	WindowCoord getWidth();
Descripción:	Retorna el valor del atributo width heredado de la clase PositionableImpl
Nombre:	void setWidth(WindowCoord)
Descripción:	Asigna un nuevo valor al atributo width

Nombre:	WindowCoord getHeight()
Descripción:	Retorna el valor del atributo height heredado de la clase PositionableImpl.
Nombre:	void setHeight(WindowCoord)
Descripción:	Asigna un nuevo valor al atributo height.
Nombre:	void assingScreen(Screen* screen) = 0
Nombre:	Screen* getScreen( ) = 0

2.2.2 - Subsistema Cairo

La principal clase que se encuentra en este subsistema es la implementación de la interfaz *Draw::Canvas* con la biblioteca gráfica Cairo. Esta interfaz define los métodos para crear primitivas (círculo, rectángulo, línea, etc.), salidas de textos con formatos de fuente y estilo y otros métodos para la optimización del renderizado de las imágenes SVG y PNG.

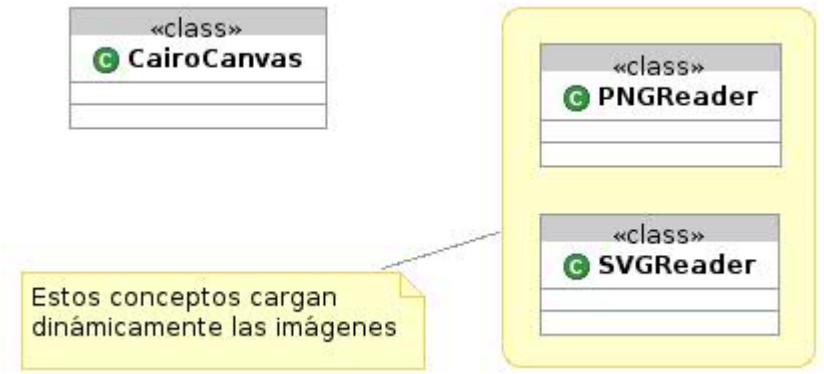

Figura 8: Diagrama de clases. Subsistema Cairo.

Nombre: CairoCanvas
Descripción: Es la implementación concreta de la clase Draw::Canvas utilizando la

biblioteca gráfica Cairo.	
Tipo de clase: Controladora	
Atributo	**Tipo**
cairoContext	cairo_t*
lineWidth	WindowCoord
fillColor	RGBAColor*
backgroundColor	RGBAColor*
lineColor	RGBAColor*
fontVariant	_cairo_font_slant
fontWeight	_cairo_font_weight
fontFace	FontFace
Para cada responsabilidad:	
Nombre:	void moveTo(Point* point)
Descripción:	Coloca el cursor en la posición inicial a partir de la cual se trazaran las lineas.
Nombre:	void lineTo(Point* point)
Descripción:	Traza una linea desde la posición inicial al punto pasado por parámetro.
Nombre:	void startPolyline( )
Descripción:	Inicializa el color con el que se trazara una polilinea.
Nombre:	void PolylineTo(Graphics::Point* point)
Descripción:	Traza una linea hacia el punto, el conjunto de estos trazos conforman la polilinea.
Nombre:	void endPolyline( )
Descripción:	Realiza el trazo de la polilinea.
Nombre:	void setLineWidth(WindowCoord width)
Descripción:	Establece el ancho con el que se realizaran los trazos.
Nombre:	void rectangle(Point* point1, Point* point2)
Descripción:	Dibuja un rectángulo con un color de relleno previamente establecido.

Nommbre:	void textOut(Text text)
Descripción:	Dibuja un texto pasado por parametros a la función.
Nombre:	void setTextSize(Size size)
Descripción:	Asigna el tamaño de la fuente tipográfica.
Nombre:	void setTextFontType(FontFace face)
Descripción:	Asigna el tipo de letra de la fuente tipográfica.
Nombre:	void setTextVariant(FontVariant variant)
Descripción:	Asigna la forma de la fuente tipográfica
Nombre:	void setTextWeight(FontWeight weight)
Descripción:	Asigna el ancho de los caracteres de la fuente tipográfica.
Nombre:	void setFillColor(RGBAColor* color)
Descripción:	Asigna un color de relleno.
Nombre:	void setLineColor(RGBAColor* color)
Descripción:	Asigna un color de la linea.
Nombre:	void save( )
Descripción:	Copia el estado actual del contexto en una pila.
Nombre:	void restore( )
Descripción:	Restaura el contexto salvado.
Nombre:	void translate(WindowCoord x, WindowCoord y)
Descripción:	Traslada el punto de origen de la matriz de transformación
Nombre:	void scale(ScaleAmount x, ScaleAmount y)
Descripción:	Escala la matriz de transformación multiplicándola por el factor de escala.
Nombre:	void rotate(Angle angle)
Descripción:	Aplica un ángulo re rotación a la matriz de transformación.
Nombre:	void ellipse(Point* point1, Point *point2)
Descripción:	Crea una elipse con los dos puntos entrados como parámetros.
Nombre:	void polygon(PointCollection points)

Descripción:	Crea un polígono con la colección de puntos.
Nombre:	void arc(Point* point,double radius,double angle1,double angle2)
Descripción:	Crea un arco circular.
Nombre:	void bitmapOut(Bitmap bitmap, WindowCoord width, WindowCoord height)
Descripción:	Dibuja una superficie con la información de bitmap y de tamaño definido por width y height.
Nombre:	void vectorTransform(double* x, double* y)
Descripción:	Transforma el espacio de coordenadas del usuario al del dispositivo.
Nombre:	void vectorInverseTransform(double* x, double* y)
Descripción:	Transforma el espacio de coordenadas del dispositivo al del usuario.
Nombre:	cairo_t* getCairoContext( )
Descripción:	Retorna el valor del atributo cairoContext.
Nombre:	void setCairoContext(cairo_t* cairoContext)
Descripción:	Asigna un nuevo valor al atributo cairoContext.
Nombre:	WindowCoord& getLineWidth( )
Descripción:	Retorna el valor del atributo lineWidth.
Nombre:	RGBAColor* getFillColor( )
Descripción:	Retorna el valor del atributo fillColor.
Nombre:	RGBAColor* getBackgroundColor( )
Descripción:	Retorna el valor del atributo backgroundColor.
Nombre:	void setBackgroundColor(RGBAColor* backgroundColor)
Descripción:	Asigna un nuevo valor al atributo backgroundColor.
Nombre:	RGBAColor* getLineColor( )
Descripción:	Retorna el valor del atributo lineColor.
Nombre:	FontFace& getFontFace( )
Descripción:	Retorna el valor del atributo fontFace.
Nombre:	void setFontFace(FontFace fontFace)

Descripción:	Asigna un nuevo valor al atributo fontFace.
Nombre:	void pixelSetOut(PixelSet& pixelSet)
Descripción:	Crea una superficie con la información de pixelSet
Nombre:	void vectorImageOut(VectorImage& vectorImage)
Descripción:	Crea una superficie con la información de vectorImage.
Nombre:	void mask(Graphics::Point& topLeft, Graphics::WindowCoord width, Graphics::WindowCoord height)
Descripción:	Crea una mascara rectángular.
Nombre:	void mask(Graphics::BoundingRect* boundingRect)
Descripción:	Crea una mascada con los puntos del boundingRect.
Nombre:	void getCTM(Draw::TransformationMatrix* matrix)
Descripción:	Guarda el estado actual de la matriz de transformación.
Nombre:	void setCTM(Draw::TransformationMatrix* matrix)
Descripción:	Adiciona una transformación a la matriz de transformación.

Nombre: PNGReader	
Descripción: Carga la información de un fichero en un buffer de memoria.	
Tipo de clase: Controladora	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	void Load(Utils::Path filename, Graphics::PixelSet& pixelSet)
Descripción:	Capturar los datos que conforman una imagen PNG, el ancho, el largo y la coloca en el bufer pixelSet.

Nombre: SVGReader
Descripción: Carga la información de un fichero en un buffer de memoria.
Tipo de clase: Controladora

Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	void Load(String filename, Graphics::VectorImage& image)
Descripción:	Parcea una imagen SVG, toma los datos, el ancho, el largo y la coloca en el bufer image.

2.2.3 - Subsistema Draw

En este subsistema se encuentran las clases con las que se construyen los objetos que se representaran en el visualizador (mayormente imágenes como botones, ventiladores, termómetros y otros dispositivos), también se encuentra la implementación del histograma con la biblioteca gráfica Cairo (clase Histogram). Se definen funcionalidades importantes heredadas del subsistema Base.

El primer diagrama muestra las relaciones entre la interfaz principal del subsistema (Drawable) y el resto de las clases definidas en el mismo y el segundo describe como se relaciona la interfaz Canvas con las clases del subsistema Draw y el mecanismo de notificación para emitir la señal de “repintado” en el subsistema.

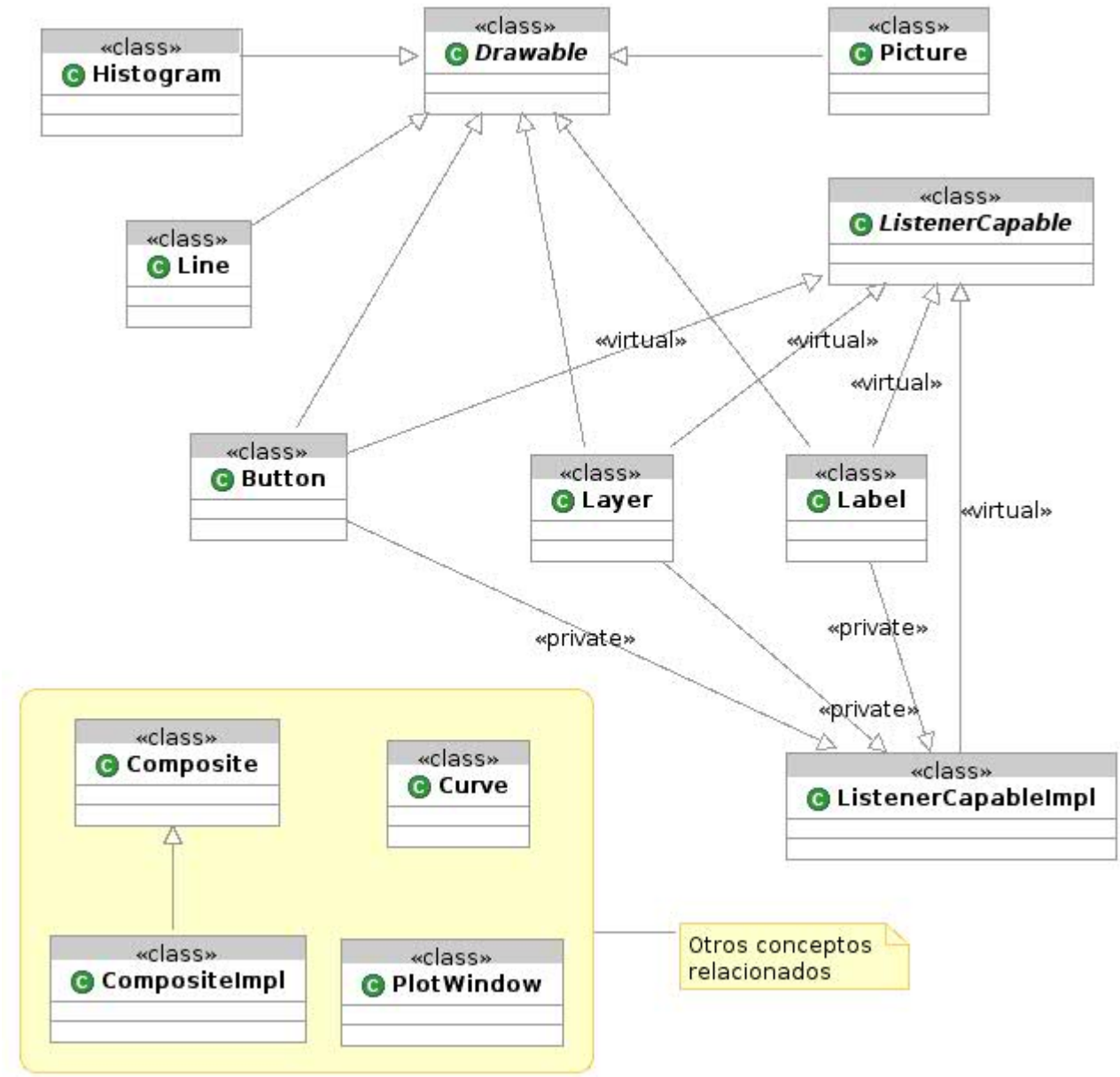

Figura 9: Diagrama de clases. Subsistema Draw 01.

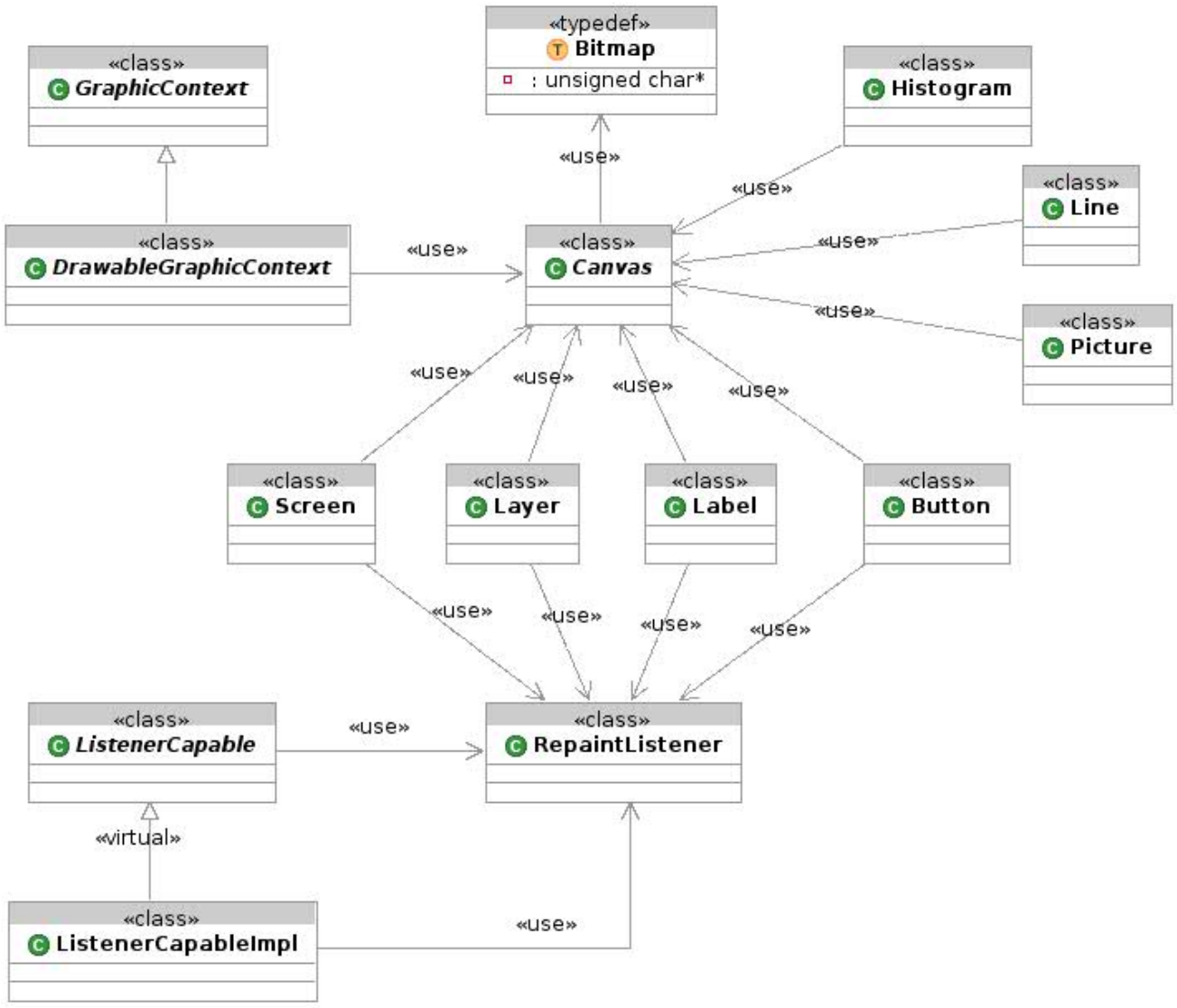

Figura 10: Diagrama de clases. Subsistema Draw 02.

Nombre: Button
Descripción: Implementa el comportamiento de un botón a partir de dos imágenes que se alternan. Hereda de las clases SCADA::HMI::Graphics::Button, Drawable,ListenerCapable, ListenerCapableImpl
Tipo de clase: Controladora

Atributo	**Tipo**
boundingRect	BoundingRect
activatedVectorImage	Graphics::VectorImage
unactivatedVectorImage	Graphics::VectorImage
activatedImage	Path
unactivatedImage	Path
Para cada responsabilidad:	
Nombre:	void setActivatedImage(Path image)
Descripción:	Asigan un nuevo valor al atributo activatedVectorImage
Nombre:	void setUnactivatedImage(Path image)
Descripción:	Asigan un nuevo valor al atributo unactivatedVectorImage
Nombre:	void draw(Canvas* canvas)
Descripción:	Implementa la capacidad de ser dibujable sobre el Canvas.
Nombre:	void notify(Event* event)
Descripción:	Implementa la capacidad de recibir eventos.
Nombre:	RepaintListener **getListener()
Descripción:	Retorna el valor del atributo heredado repaintListener.
Nombre:	void setListener(Draw::RepaintListener **repaintListener)
Descripción:	Asigna un nuevo valor al atributo heredado repaintListener

Nombre: Canvas	
Descripción: Interfaz que provee los métodos para dibujar en el Canvas	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual void ellipse(Point* point1, Point *point2) = 0
Nombre:	virtual void moveTo(Graphics::Point* point) = 0

Nombre:	virtual void lineTo(Graphics::Point* point) = 0
Nombre:	virtual void startPolyline() = 0
Nombre:	virtual void PolylineTo(Graphics::Point* point) = 0
Nombre:	virtual void endPolyline() = 0
Nombre:	virtual void setLineWidth(Graphics::WindowCoord width) = 0
Nombre:	virtual void rectangle(Graphics::Point* point1, Point* point2) = 0
Nombre:	virtual void textOut(Text text) = 0
Nombre:	virtual void setTextSize(Size size) = 0
Nombre:	virtual void setTextFontType(FontFace face) = 0
Nombre:	virtual void setTextVariant(FontVariant variant) = 0
Nombre:	virtual void setTextWeight(FontWeight weight) = 0
Nombre:	virtual void setFillColor(RGBAColor* color) = 0
Nombre:	virtual void setLineColor(RGBAColor* color) = 0
Nombre:	virtual void save() = 0
Nombre:	virtual void restore() = 0
Nombre:	virtual void translate(WindowCoord x, WindowCoord y) = 0
Nombre:	virtual void scale(ScaleAmount x, ScaleAmount y) = 0
Nombre:	virtual void rotate(Angle angle) = 0
Nombre:	virtual void ellipse(Point* point1, Point *point2) = 0
Nombre:	virtual void polygon(PointCollection points) = 0
Nombre:	virtual void arc(Point* point, double radius, double angle1, double angle2) = 0
Nombre:	virtual void bitmapOut(Bitmap bitmap, WindowCoord width, WindowCoord height) = 0
Nombre:	virtual void vectorTransform(double* x, double* y) = 0
Nombre:	virtual void vectorInverseTransform(double* x, double* y) = 0
Nombre:	virtual void pixelSetOut(Graphics::PixelSet& pixelSet) = 0
Nombre:	virtual void vectorImageOut(Graphics::VectorImage& vectorImage) = 0

Nombre:	virtual void mask(Graphics::Point& topLeft, Graphics::WindowCoord width, Graphics::WindowCoord height) = 0
Nombre:	virtual void mask(Graphics::BoundingRect* boundingRect) = 0
Nombre:	virtual void getCTM(TransformationMatrix* matrix) = 0
Nombre:	virtual void setCTM(TransformationMatrix* matrix) = 0

Nombre: Composite	
Descripción: Interfaz que deben implementar las clases que sean compuestas por objetos dibujables mas simples.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual const Collection& getDrawables() const = 0

Nombre: Curve	
Descripción: Clase que implementa las curvas en el que se dibujaran en el histograma.	
Tipo de clase: Controladora	
Atributo	**Tipo**
lineStyle	LineStyle
lineColor	RGBAColor*
max	VariableValue
min	VariableValue
variableSet	VariableSet
Para cada responsabilidad:	
Nombre:	LineStyle getLineStyle()
Descripción:	Retorna el valor del atributo lineStyle.

Nombre:	void setLineStyle(LineStyle lineStyle)
Descripción:	Asigana un nuevo valor al atributo lineStyle.
Nombre:	RGBAColor* getLineColor ()
Descripción:	Retorna el valor del atributo lineColor.
Nombre:	void setLineColor(RGBAColor* lineColor)
Descripción:	Asigna un nuevo valor al atributo lineColor.
Nombre:	VariableValue getMaxValue ()
Descripción:	Retorna el valor del atributo maxValue.
Nombre:	void setMaxValue(VariableValue max)
Descripción:	Asigna un nuevo valor al atributo max
Nombre:	VariableValue getMinValue ()
Descripción:	Retorna el valor del atributo minValue.
Nombre:	void setMinValue(VariableValue min)
Descripción:	Asigna un nuevo valor al atributo min.
Nombre:	VariableSet& getVariableSet ( )
Descripción:	Retorna el valor del atributo variableSet.
Nombre:	void setVariables(VariableSet variablesSet)
Descripción:	Asigna un nuevo valor al atributo variablesSet

Nombre: Drawable	
Descripción: Interfaz que define la capacidad de un objeto de ser dibujable en el Canvas	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual void draw(Canvas* canvas) = 0
Nombre:	virtual void redraw(Canvas* canvas) = 0

Nombre: Histogram	
Descripción: Dibujar en el Canvas el histograma con las curvas y los atributos. Hereda de las clases SCADA::HMI::Graphics::Histogram, Drawable.	
Tipo de clase: Controladora	
Atributo	**Tipo**
backgroundColor	RGBAColor*
plotAreaBackgroundColor	RGBAColor*
axisXLabel	HistogramString
axisYLabel	HistogramString
title	HistogramString
axisColor	RGBAColor*
windowTopLeft	Graphics::Point
windowWidth	Graphics::WindowCoord
windowHeight	Graphics::WindowCoord
windowOffset	Graphics::WindowCoord
division	unsigned int
curves	CurveCollection
Para cada responsabilidad:	
Nombre:	RGBAColor* getBackgroundColor ( )
Descripción:	Retorna el valor del atributo backgroundColor.
Nombre:	void setBackgroundColor(RGBAColor* color)
Descripción:	Asigna un nuevo valor al atributo backgroundColor
Nombre:	HistogramString getAxisXLabel ( )
Descripción:	Retorna el valor del atributo axisXLabel.
Nombre:	void setAxisXLabel(HistogramString axisXLabel)
Descripción:	Asigna un nuevo valor al atributo axisXLabel.
Nombre:	HistogramString getAxisYLabel ( )

Descripción:	Retorna el valor del atributo axisYLabel.
Nombre:	void setAxisYLabel(HistogramString axisYLabel)
Descripción	Asigna un nuevo valor al atributo axisYLable.
Nombre:	HistogramString getHistogramTitle ( )
Descripción:	Retorna el valor del atributo histogramTitle.
Nombre:	void setHistogramTitle(HistogramString title)
Descripción:	Asigna un nuevo valor al atributo histogramTitle.
Nombre:	RGBAColor* getAxiscolor( )
Descripción:	Retorna el valor del atributo axisColor.
Nombre:	void setAxisColor(RGBAColor* axisColor)
Descripción:	Asigna un nuevo valor al atributo axisColor.
Nombre:	void draw (Canvas* canvas)
Descripción:	Implementa como el histograma se va a pintar en el Canvas mediante funciones proporcionadas por la librería Cairo.

Nombre: Label	
Descripción: Representa una etiqueta o texto en el Canvas. Hereda de las clases UpdateableControl, Graphics::Positionable, Graphics::PositionableImpl	
Tipo de clase: Controladora	
Atributo	**Tipo**
text	String
Para cada responsabilidad:	
Nombre:	String getText()
Descripción:	Retorna el valor del atributo text
Nombre:	void setText(String text)
Descripción:	Asigna un nuevo valor al atributo text
Nombre:	Point* getTopLeft ()
Descripción:	Retorna el valor del atributo heredado topLeft.

Nombre:	void setTopLeft (Point* topLeft)
Descripción:	Asigna un nuevo valor al atributo topLeft.
Nombre:	WindowCoord getWidth()
Descripción:	Retorna el valor del atributo heredado width
Nombre:	void setWidth(WindowCoord)
Descripción:	Asigna un nuevo valor al atributo width
Nombre:	WindowCoord getHeight()
Descripción:	Retorna el valor del atributo heredado height.
Nombre:	void setHeight(WindowCoord)
Descripción:	Asigna un nuevo valor al atributo height.

Nombre: Layer	
Descripción: Representa una capa que sera asignada a un Screen. Hereda de las clases SCADA::HMI::Graphics::Layer, Drawable, Notifiable,ListenerCapable, ListenerCapableImpl.	
Tipo de clase: Controladora	
Atributo	**Tipo**
dispatcher	EventDispatcher
graphics	DrawableGraphicCollection
Para cada responsabilidad:	
Nombre:	void addGraphicObject(SCADA::HMI::Graphics::Graphic* object)
Descripción:	Añadir a una lista los objetos que seran dibujados en el Canvas.
Nombre:	GraphicCollection& getGraphicObjectCollection( )
Descripción:	Retorna la colección de objetos gráficos.
Nombre:	void draw(Canvas* canvas)
Descripción:	Implementa como se dibujan los objetos gráficos de la lista.
Nombre:	virtual void notify(Event* event)
Descripción:	Notifica los eventos que se generan.

Nombre:	RepaintListener **getListener()
Descripción:	Retorna el valor del atributo repaintListener.
Nombre:	void setListener(Draw::RepaintListener **repaintListener)
Descripción:	Asigna un nuevo valor al atributo repaintListener

Nombre: Line	
Descripción: Es una linea que se dibujara en el Canvas. Hereda de las clases SCADA::HMI::Graphics::Line, Drawable	
Tipo de clase: Controladora	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual void draw(Canvas* canvas)
Descripción:	Implementa como se dibuja la linea.

Nombre: ListenerCapable	
Descripción: Interzas que implementas todos los objetos que recibirán eventos.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual void setListener(RepaintListener **repaintListener) = 0
Nombre:	virtual RepaintListener** getListener( ) = 0

Nombre: ListenerCapableImpl
Descripción: Clase que implementa el comportamiento de los objetos que tendrán la capacidad de escuchar eventos.

Tipo de clase: Controladora	
Atributo	**Tipo**
repaintListener	RepaintListener**
Para cada responsabilidad:	
Nombre:	virtual RepaintListener** getListener( )
Descripción:	Retorna el valor del atributo repaintListener
Nombre:	virtual void setListener(RepaintListener **repaintListener)
Descripción:	Establece un nuevo valor para el atributo repaintListener que es pasado como parámetro.

Nombre: Picture	
Descripción: Es la implementación concreta de Graphics::Picture	
Tipo de clase: Controladora	
Atributo	**Tipo**
isPNG	bool
fileName	Path
vectorImage	Graphics::VectorImage
pixelSet	Graphics::PixelSet
Para cada responsabilidad:	
Nombre:	virtual void loadSVG(Path fileName)
Descripción:	Carga las imágenes con formato SVG
Nombre:	virtual void loadPNG(Path fileName)
Descripción:	Carga las imágenes con formato PNG.
Nombre:	virtual void draw(Canvas* canvas)
Descripción:	Implementa la manera de mostrar las imágenes.

Nombre: RepaintListener
Descripción: Se encarga de estar a la escucha de los eventos de repintado para los

objetos.	
Tipo de clase: Controladora	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual void repaintRequest(Graphics::BoundingRect* boundRect)
Descripción:	Pasa un evento de repintado que se genera dentro del boundRect.
Nombre:	virtual void repaintRequest(Graphics::BoundingRect* boundRect, Draw::Drawable* object)
Descripción:	Pasa un evento de repintado que se genera dentro del boundRect a un objeto determinado.

Nombre: Screen	
Descripción: Es la implementación concreta de la clase Base::Screen. Hereda de las clases Base::Screen y Notifiable.	
Tipo de clase: Controladora	
Atributo	**Tipo**
repaintListener	Draw::RepaintListener*
screenListener	GTK::GTKDrawableAsyncronusListener*
dispatcher	Events::EventDispatcher
layers	Graphics::LayerCollection
Para cada responsabilidad:	
Nombre:	void notify(Events::Event* event)
Descripción:	Notifica los eventos que se producen en el screen.
Nombre:	void addLayer(Graphics::Layer* layer)
Descripción:	Añade una capa a la lista de capas que se representaran.
Nombre:	Graphics::LayerCollection& getLayerCollection()
Descripción:	Retorna los datos del atributo layerCollection.

Nombre:	void draw(Draw::Canvas* canvas)
Descripción:	Dibuja la colección de capas en el canvas.
Nombre:	RepaintListener **getListener()
Descripción:	Retorna el valor del atributo repaintListener.
Nombre:	void setListener(Draw::RepaintListener **repaintListener)
Descripción:	Asigna un nuevo valor al atributo repaintListener

2.2.4 - Subsistema Graphics

En el subsistema Graphics se definen los principales objetos gráficos que se mostrarán en los diferentes despliegues. Las clases se agruparon dependiendo de los elementos que tenían en común. El primero define las características de todos los objetos gráficos en general, el segundo diagrama define los las particularidades de las formas básicas en los despliegues, el tercer diagrama de clases representa de manera muy sencilla los detalles de los objetos de tipo imagen o figura. Por último se definen los “controles” en el despliuegue, estos últimos con su particularidad de cambiar su estado mediante un mecanismo de actualización.

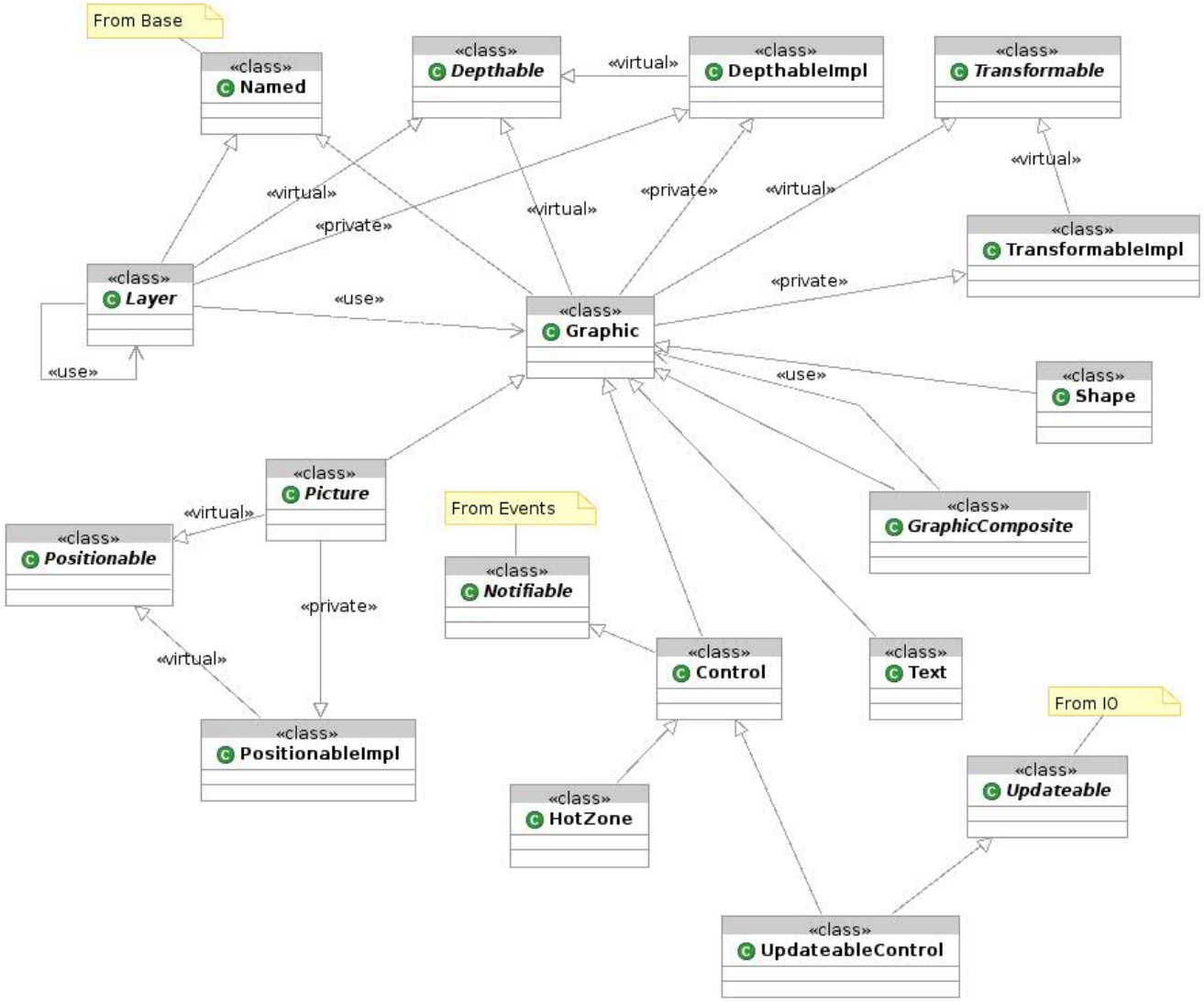

Figura 11: Diagrama de clases. Subsistema Graphics 01.

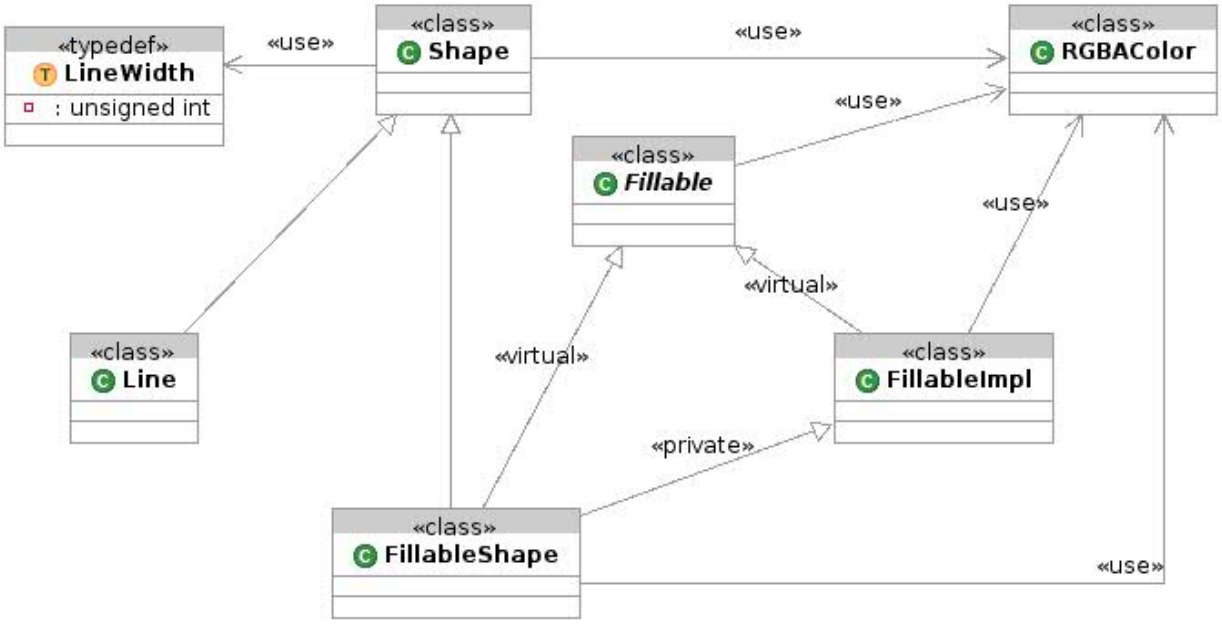

Figura 12: Diagrama de clases. Subsistema Graphics 02.

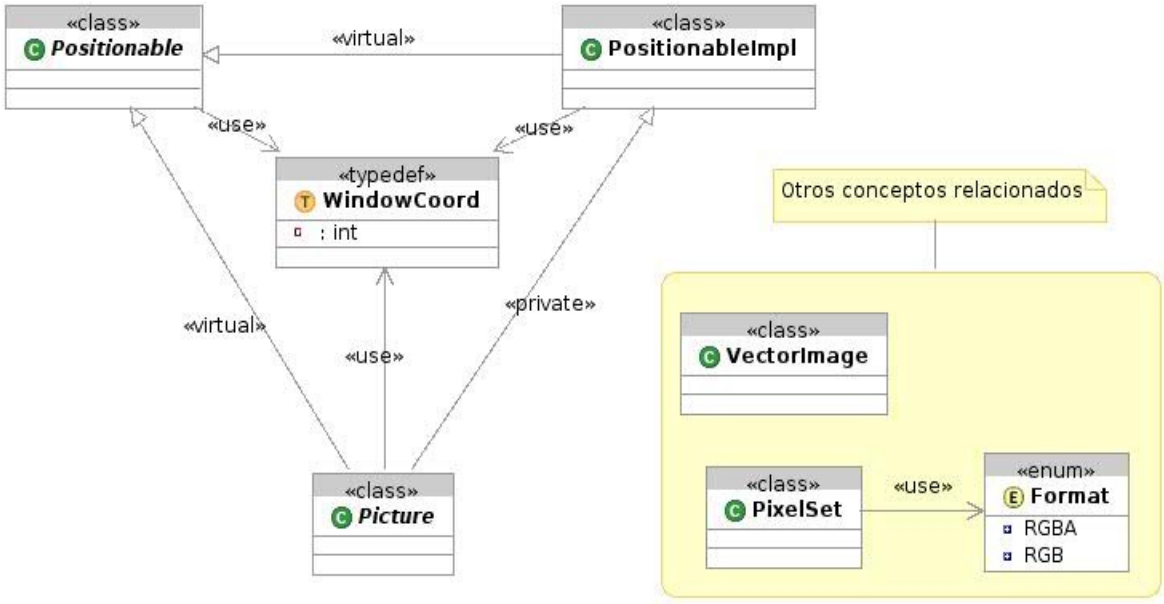

Figura 13: Diagrama de clases. Subsistema Graphics 03.

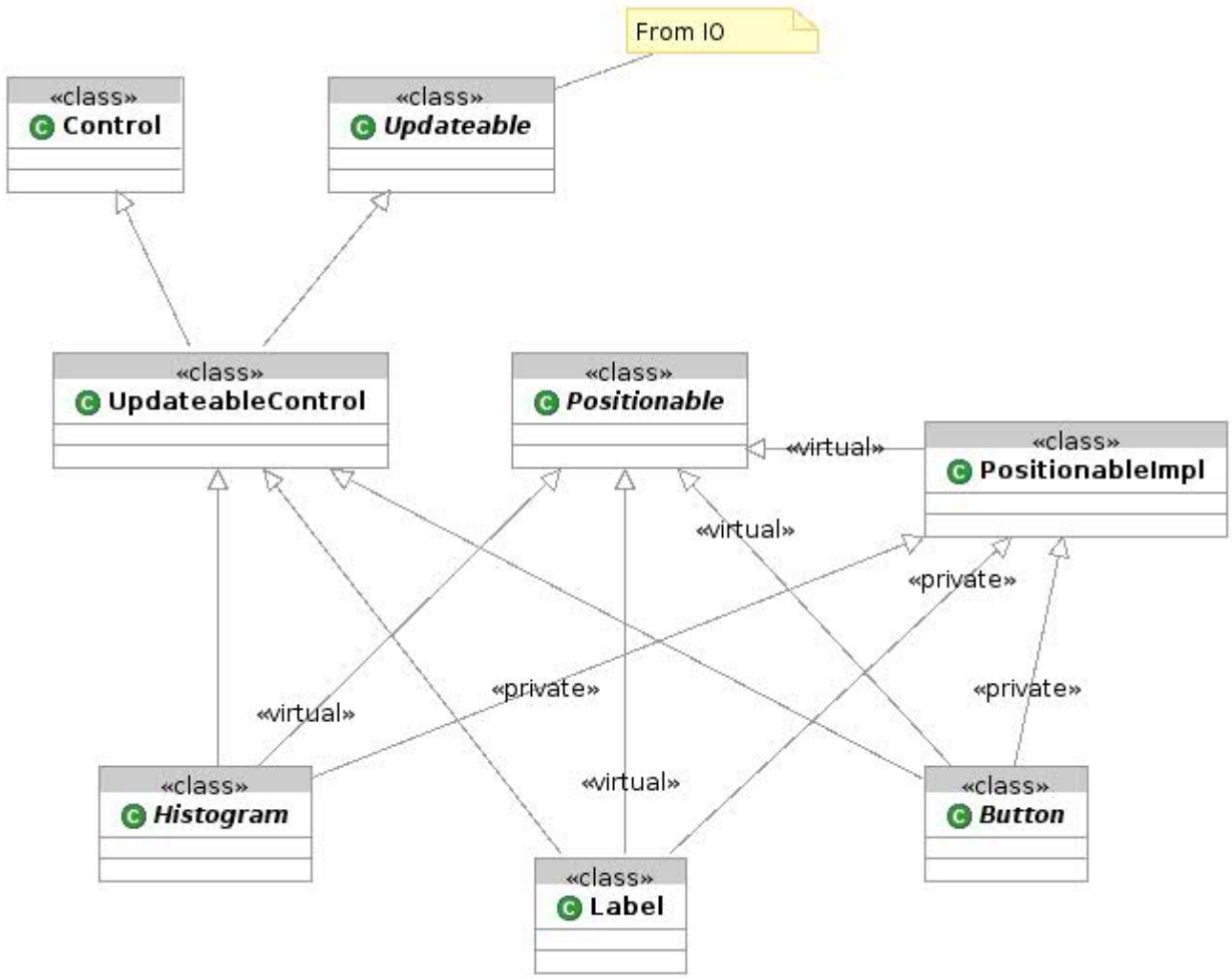

Figura 14: Diagrama de clases. Subsistema Graphics 04.

Nombre: Dephtable	
Descripción: Interfaz que define que un objeto tiene profundidad.	
Tipo de clase: Interfaz	
Atributo	**Tipo**

Para cada responsabilidad:	
Nombre:	Depth getDepth()
Nombre:	void setDepth(Depth depth)

Nombre: DephtableImpl		
Descripción: Clase que implementa el comportamiento común de los objetos que tienen profundidad. Hereda de la clase: Dephtable.		
Tipo de clase: Interfaz		
Atributo		**Tipo**
depth		Depth
Para cada responsabilidad:		
Nombre:	Depth getDepth()	
Descripción:	Retorna el valor del atributo depth.	
Nombre:	void setDepth()	
Descripción:	Asigna un nuevo valor del atributo depth.	

Nombre: Fillable	
Descripción: Interfaz que define que un objeto se rellena con un color.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual RGBAColor* getFillColor() const
Nombre:	virtual void setFillColor(RGBAColor* fillColor)

Nombre: FillableImpl	
Descripción: Clase que implementa el comportamiento común de los objetos que tienen un color de fondo. Hereda de la clase: Fillable.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
fillColor	RGBAColor
Para cada responsabilidad:	
Nombre:	virtual RGBAColor* getFillColor() const
Descripción:	Retorna el valor del atributo fillColor.
Nombre:	virtual void setFillColor(RGBAColor* fillColor)
Descripción:	Asigna un nuevo valor del atributo fillColor.

Nombre: Positionable	
Descripción: Interfaz que define que un objeto puede ubicarse en un punto del despliegue.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual Point* getTopLeft() = 0
Nombre:	virtual WindowCoord getWidth() = 0
Nombre:	virtual WindowCoord getHeight() = 0
Nombre:	virtual void setTopLeft(Point* topLeft) = 0
Nombre:	virtual void setWidth(WindowCoord) = 0
Nombre:	virtual void setHeight(WindowCoord) = 0

Nombre: PositionableImpl
Descripción: Clase que implementa el comportamiento común de los objetos que se

pueden ubicar en un punto del despliegue. Hereda de la clase: Positionable.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
topLeft	Point
width	WindowCoord
heigth	WindowCoord
Para cada responsabilidad:	
Nombre:	virtual Point* getTopLeft()
Descripción:	Retorna el valor del atributo topLeft.
Nombre:	virtual WindowCoord getWidth()
Descripción:	Retorna el valor del atributo width.
Nombre:	virtual WindowCoord getHeight()
Descripción:	Retorna el valor del atributo heigth.
Nombre:	virtual void setTopLeft(Point* topLeft)
Descripción:	Asigna un nuevo valor al atributo topLeft.
Nombre:	virtual void setWidth(WindowCoord width)
Descripción:	Asigna un nuevo valor al atributo width.
Nombre:	virtual void setHeight(WindowCoord heigth)
Descripción:	Asigna un nuevo valor al atributo heigth.

Nombre: Transformable	
Descripción: Interfaz que define que un objeto puede realizar operaciones de transformación (rotar, escalar, mover).	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	

Nombre:	virtual ScaleVector& getScaleVector() = 0
Nombre:	virtual void traslate(Point* traslateVectorPoint) = 0
Nombre:	virtual void rotate(Angle angle) = 0
Nombre:	virtual void setRotationAxis (Point* rotationPoint) = 0
Nombre:	virtual void scale(ScaleVector& scaleVector) = 0

Nombre: TransformableImpl	
Descripción: Clase que implementa el comportamiento común de los objetos que pueden sufrir transformaciones. Hereda de la clase: Transformable.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
scaleVector	ScaleVector
traslatePoint	Point
rotationAxisPoint	Point
rotationAngle	Angle
Para cada responsabilidad:	
Nombre:	virtual ScaleVector& getScaleVector()
Descripción:	Retorna el valor del atributo scaleVector.
Nombre:	virtual void scale(ScaleVector& scaleVector)
Descripción:	Escala el objeto utilizando el vector de escala scaleVector.
Nombre:	virtual void traslate(Point* traslateVectorPoint)
Descripción:	Traslada el objeto hacia el punto traslateVectorPoint.
Nombre:	virtual void rotate(Angle angle)
Descripción:	Rota el objeto en angle grados.
Nombre:	virtual void setRotationAxis (Point* rotationPoint)
Descripción:	Asigna un nuevo punto por donde pasa el eje de rotación.

Nombre: RGBAColor	
Descripción: Clase que define a un color de tipo Red Blue Green Alpha.	
Tipo de clase: Entidad	
Atributo	**Tipo**
red	int
green	int
blue	int
alpha	int
Para cada responsabilidad:	
Nombre:	ColorDimension getRed()
Descripción:	Retorna el valor del atributo red.
Nombre:	ColorDimension getGreen()
Descripción:	Retorna el valor del atributo green.
Nombre:	ColorDimension getBlue()
Descripción:	Retorna el valor del atributo blue.
Nombre:	ColorDimension getAlpha()
Descripción:	Retorna el valor del atributo alpha.
Nombre:	void setRed(ColorDimension red)
Descripción:	Asigna un nuevo valor al atributo red.
Nombre:	void setGreen(ColorDimension green)
Descripción:	Asigna un nuevo valor al atributo green.
Nombre:	void setBlue(ColorDimension blue)
Descripción:	Asigna un nuevo valor al atributo blue.
Nombre:	void setAlpha(ColorDimension alpha)
Descripción:	Asigna un nuevo valor al atributo alpha.
Nombre:	void operator=(RGBAColor *color)
Descripción:	Copia los valores de color hacia los de la clase.
Nombre:	void operator=(RGBAColor& color)

Descripción:	Copia los valores de color hacia los de la clase.
Nombre: Point	
Descripción: Clase que define a un punto en el despliegue.	
Tipo de clase: Entidad	
Atributo	**Tipo**
x	WindowCoord
y	WindowCoord
Para cada responsabilidad:	
Nombre:	WindowCoord getX()
Descripción:	Retorna el valor del atributo x.
Nombre:	WindowCoord getY()
Descripción:	Retorna el valor del atributo y.
Nombre:	void setX(WindowCoord x)
Descripción:	Asigna un nuevo valor al atributo x.
Nombre:	void setY(WindowCoord y)
Descripción:	Asigna un nuevo valor al atributo y.
Nombre:	void operator=(Point* point)
Descripción:	Copia los valores de point hacia los de la clase.
Nombre:	void operator=(Point& point)
Descripción:	Copia los valores de point hacia los de la clase.

Nombre: Graphic	
Descripción: Es la clase de la que heredan los objetos que se muestran en el despliegue. Herada de las clases: Named, Dephtable, DephtableImpl, Transformable, TransformableImpl.	
Tipo de clase: Entidad	
Atributo	**Tipo**

Para cada responsabilidad:	
Nombre:	Depth getDepth()
Descripción:	Retorna el valor del atributo depth.
Nombre:	ScaleVector& getScaleVector()
Descripción:	Retorna el vector escalado despues de aplicar alguna transformación.
Nombre:	static bool graphicSortPredicate(Graphic* leftOperand, Graphic* rigthOperand)
Descripción:	Determina si la profundidad de un operando es menor que la del otro, se usa para determinar qué objeto debe dibujarse sobre el otro de acuerdo a la profundidad.
Nombre:	void setDepth(Depth depth)
Descripción:	Asigna un nuevo valor al atributo depth.
Nombre:	void scale(ScaleVector& scaleVector)
Descripción:	Escala el gráfico usando el vector de escala scaleVector.
Nombre:	void traslate(Point* traslateVectorPoint)
Descripción:	Traslada el gráfico hacia el punto traslateVectorPoint.
Nombre:	void rotate(Angle angle)
Descripción:	Rota el gráfico la cantidad de grados definidos por angle.
Nombre:	void setRotationAxis (Point* rotationPoint)
Descripción:	Asigna un nuevo valor al punto guía del eje de rotación.

Nombre: GraphicComposite	
Descripción: Clase que implementa el patrón Composite para los objetos de tipo Graphic. Hereda de la clase: Graphic.	
Tipo de clase: Interfaz	
Atributo	**Tipo**

Para cada responsabilidad:	
Nombre:	virtual void addChild(Graphic* child) = 0
Nombre:	virtual GraphicCollection& getChildList() = 0

Nombre: Shape	
Descripción: Clase que define a una “forma” en el despliegue.	
Tipo de clase: Entidad	
Atributo	**Tipo**
lineWidth	LineWidth
lineColor	RGBAColor
Para cada responsabilidad:	
Nombre:	virtual const LineWidth getLineWidth()
Descripción:	Retorna el valor del atributo lineWidth.
Nombre:	virtual RGBAColor* getLineColor()
Descripción:	Retorna el valor del atributo lineColor.
Nombre:	virtual void setLineWidth(LineWidth width)
Descripción:	Asigna un nuevo valor al atributo lineWidth.
Nombre:	virtual void setLineColor(RGBAColor* color)
Descripción:	Retorna el valor del atributo lineColor.

Nombre: FillableShape	
Descripción: Clase que define a todas las formas que que tienen color de relleno.	
Tipo de clase: Entidad	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	RGBAColor* getFillColor() const

Descripción:	Retorna el valor del atributo fillColor.
Nombre:	void setFillColor(RGBAColor* fillColor)
Descripción:	Retorna el valor del atributo fillColor.

Nombre: Label		
Descripción: Clase que representa a una etiqueta en el despliegue. Hereda de las clases: UpdateableControl, Positionable, PositionableImpl.		
Tipo de clase: Entidad		
Atributo	**Tipo**	
text	String	
Para cada responsabilidad:		
Nombre:	String getText() const	
Descripción:	Retorna el valor del atributo text.	
Nombre:	Point* getTopLeft ()	
Descripción:	Retorna el valor del atributo topLeft.	
Nombre:	WindowCoord getWidth()	
Descripción:	Retorna el valor del atributo width.	
Nombre:	WindowCoord getHeight()	
Descripción:	Retorna el valor del atributo heigth.	
Nombre:	void setText(String text)	
Descripción:	Asigna un nuevo valor al atributo text.	
Nombre:	void setTopLeft (Point* topLeft)	
Descripción:	Asigna un nuevo valor al atributo topLeft.	
Nombre:	void setWidth(WindowCoord width)	
Descripción:	Asigna un nuevo valor al atributo width.	
Nombre:	void setHeight(WindowCoord heigth)	
Descripción:	Asigna un nuevo valor al atributo heigth.	

Nombre: Layer	
Descripción: Clase que representa a una capa en el despliegue. Hereda de las clases: Named, Dephtable, DephtableImpl.	
Tipo de clase: Entidad	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	Depth getDepth ()
Descripción:	Retorna el valor del atributo depth.
Nombre:	static bool layerSortPredicate(Layer* leftOperand, Layer* rigthOperand)
Descripción:	Define si una capa tiene mayor profundidad que otra, se usa para dibujar una sobre la otra en dependencia del atributo depth.
Nombre:	virtual GraphicCollection& getGraphicObjectCollection() = 0
Nombre:	virtual void addGraphicObject(Graphic* object) = 0

Nombre: Line	
Descripción: Clase que representa a una línea (unión de dos puntos). Hereda de la clase: Shape.	
Tipo de clase: Entidad	
Atributo	**Tipo**
initialVertex	Point
finalVertex	Point
Para cada responsabilidad:	
Nombre:	virtual Point* getInitialVertex() const
Descripción:	Retorna el valor del atributo initialVertex.
Nombre:	virtual Point* getFinalVertex() const
Descripción:	Retorna el valor del atributo initialVertex.

Nombre:	virtual void setInitialVertex(Point* initialVertex)
Descripción:	Asigna un nuevo valor al atributo initialVertex.
Nombre:	virtual void setFinalVertex(Point* finalVertex)
Descripción:	Asigna un nuevo valor al atributo finalVertex.

Nombre: Button	
Descripción: Clase que modela el comportamiento de un botón. Hereda de las clases: UpdateableControl, Positionable, PositionableImpl.	
Tipo de clase: Entidad	
Atributo	**Tipo**
activated	bool
Para cada responsabilidad:	
Nombre:	virtual Point* getTopLeft()
Descripción:	Retorna el valor del atributo topLeft.
Nombre:	virtual WindowCoord getWidth()
Descripción:	Retorna el valor del atributo width.
Nombre:	virtual WindowCoord getHeight()
Descripción:	Retorna el valor del atributo heigth.
Nombre:	virtual void setTopLeft(Point* topLeft)
Descripción:	Asigna un nuevo valor al atributo topLeft.
Nombre:	virtual void setWidth(WindowCoord width)
Descripción:	Asigna un nuevo valor al atributo width.
Nombre:	virtual void setHeight(WindowCoord)
Descripción:	Asigna un nuevo valor al atributo heigth.
Nombre:	virtual void setActivatedImage(Path image) = 0
Nombre:	virtual void setUnactivatedImage(Path image) = 0

Nombre: Picture	
Descripción: Clase que modela una imagen en el despliegue. Hereda de las clases: Graphic, Positionable, PositionableImpl.	
Tipo de clase: Entidad	
Atributo	**Tipo**
activated	bool
Para cada responsabilidad:	
Nombre:	Point* getTopLeft()
Descripción:	Retorna el valor del atributo topLeft.
Nombre:	WindowCoord getWidth()
Descripción:	Retorna el valor del atributo width.
Nombre:	WindowCoord getHeight()
Descripción:	Retorna el valor del atributo heigth.
Nombre:	void setTopLeft(Point* topLeft)
Descripción:	Asigna un nuevo valor al atributo topLeft.
Nombre:	void setWidth(WindowCoord width)
Descripción:	Asigna un nuevo valor al atributo width.
Nombre:	void setHeight(WindowCoord heigth)
Descripción:	Asigna un nuevo valor al atributo heigth.
Nombre:	virtual void loadSVG(Path path) = 0
Nombre:	virtual void loadPNG(Path path) = 0

Nombre: PixelSet	
Descripción: Clase que modela una imagen rasterizada en el despliegue.	
Tipo de clase: Entidad	
Atributo	**Tipo**
data	unsigned char*
width	unsigned int

heigth	unsigned int
format	Format
Para cada responsabilidad:	
Nombre:	unsigned char* getData()
Descripción:	Retorna el valor del atributo data.
Nombre:	unsigned int getWidth()
Descripción:	Retorna el valor del atributo width.
Nombre:	unsigned int getHeight()
Descripción:	Retorna el valor del atributo heigth.
Nombre:	Format getFormat()
Descripción:	Retorna el valor del atributo format.
Nombre:	void setData(unsigned char* data)
Descripción:	Asigna un nuevo valor al atributo data.
Nombre:	void setWidth(unsigned int width)
Descripción:	Asigna un nuevo valor al atributo width.
Nombre:	void setHeight(unsigned int height)
Descripción:	Asigna un nuevo valor al atributo heigth.
Nombre:	void setFormat(Format format)
Descripción:	Asigna un nuevo valor al atributo format.

Nombre: VectorImage	
Descripción: Clase que modela una imagen vectorial en el despliegue.	
Tipo de clase: Entidad	
Atributo	**Tipo**
data	SVGImage
width	unsigned int
heigth	unsigned int
Para cada responsabilidad:	

Nombre:	SVGImage getData()
Descripción:	Retorna el valor del atributo data.
Nombre:	unsigned int getWidth()
Descripción:	Retorna el valor del atributo width.
Nombre:	unsigned int getHeight()
Descripción:	Retorna el valor del atributo heigth.
Nombre:	void setData(SVGImage data)
Descripción:	Asigna un nuevo valor al atributo data.
Nombre:	void setWidth(unsigned int width)
Descripción:	Asigna un nuevo valor al atributo width.
Nombre:	void setHeight(unsigned int height)
Descripción:	Asigna un nuevo valor al atributo heigth.

Nombre: Histogram	
Descripción: Clase que define como dibujar la gráfica que contiene las curvas de tendencia. Hereda de las clases: UpdateableControl, Positionable, PositionableImpl.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual RGBAColor* getBackgroundColor ( ) = 0
Nombre:	virtual HistogramString getAxisXLabel ( ) = 0
Nombre:	virtual HistogramString getAxisYLabel ( ) = 0
Nombre:	virtual HistogramString getHistogramTitle ( ) = 0
Nombre:	virtual RGBAColor* getAxiscolor( ) = 0
Nombre:	Point* getTopLeft ()
Nombre:	WindowCoord getWidth()

Nombre:	WindowCoord getHeight()
Nombre:	virtual void setBackgroundColor(RGBAColor* color) = 0
Nombre:	virtual void setAxisXLabel(HistogramString axisXLabel) = 0
Nombre:	virtual void setAxisYLabel(HistogramString axisYLabel) = 0
Nombre:	virtual void setHistogramTitle(HistogramString title) = 0
Nombre:	virtual void setAxisColor(RGBAColor* axisColor) = 0
Nombre:	void setTopLeft (Point* topLeft)
Nombre:	virtual void setWidth(WindowCoord width)
Nombre:	virtual void setHeight(WindowCoord height)

2.2.5 - Subsistema IO

En este subsistema se modela el mecanismo de Entrada/Salida (Input/Output, I/O) del visualizador en el SCADA. En el subsistema IO se han definido un grupo de entidades de suma importancia en el sistema, ya que proveen los mecanismos para actualizar la información de los despliegues.

El elemento más significativo del subsistema es la interfaz Updateable. También es de suma importancia IOManager, encargado de manipular los datos de I/O y en particular ejecutar el caso de uso más significativo identificado en este trabajo: adquirir la información de entrada (Retrieve Input).

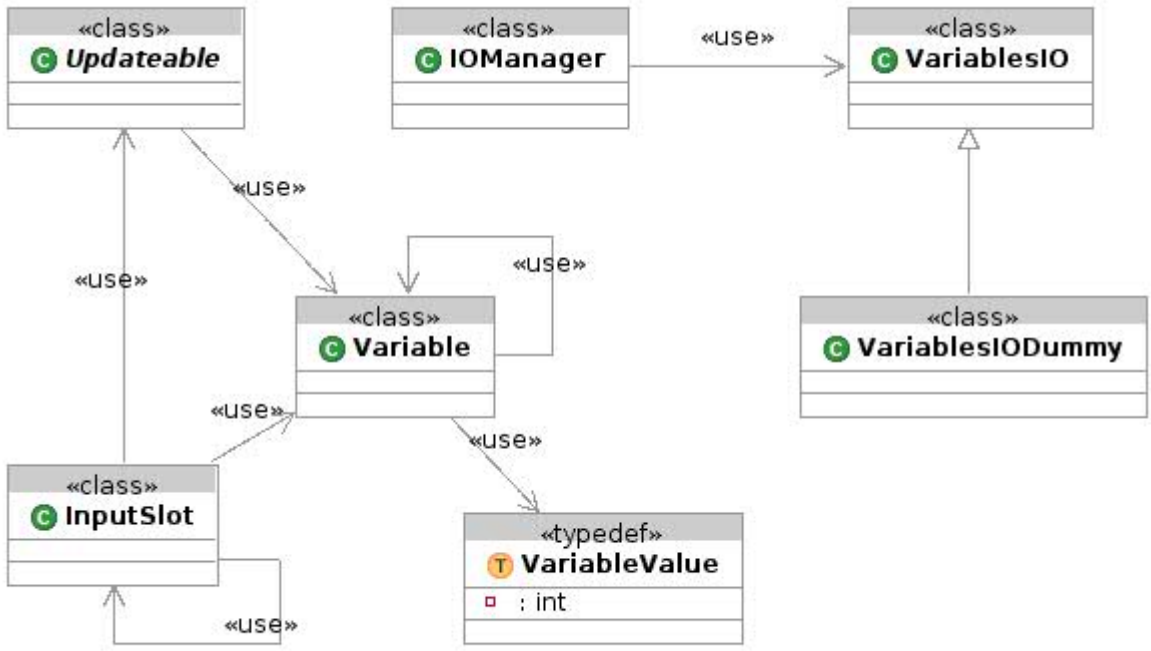

Figura 15: Diagrama de clases. Subsistema IO.

Nombre: InputSlot	
Descripción: Representa un contenedor de objetos actualizables.	
Tipo de clase: Controladora	
Atributo	**Tipo**
updateableCollection	UpdateableCollection
varId	varId
Para cada responsabilidad:	
Nombre:	void attach(Updateable* updateable)
Descripción:	Añade el elemento a la colección de elementos actualizarles.
Nombre:	void detach(Updateable* updateable)
Descripción:	Elimina el elemento de la colección de elementos actualizables.

Nombre:	void setInput(Variable* variable)
Descripción:	Actualiza la colección de variables.
Nombre:	const UpdateableCollection& getUpdateables( )
Descripción:	Retorna el valor del atributo updateableCollection.
Nombre:	VarId getVarId( )
Descripción:	Retorna el valor del atributo varId
Nombre:	void setVarId(VarId varId)
Descripción:	Asigna un nuevo valor al atributo id.

Nombre: IOManager	
Descripción: Es la clase que se encarga de administrar las entradas y las salidas de los datos y tiene la responsabilidad de manipular los Slot que contendrán los objetos.	
Tipo de clase: Controladora	
Atributo	**Tipo**
inputSlotCollection	InputSlotCollection
variablesIO	VariablesIO*
Para cada responsabilidad:	
Nombre:	void setVariablesIO(VariablesIO* variablesIO)
Descripción:	Asigna un nuevo valor al atributo variablesIO.
Nombre:	void createInputSlot(VarId id)
Descripción:	Crea un InputSlot con un número identificador y lo añade a inputSlotCollection.
Nombre:	void deleteInputSlot(VarId id)
Descripción:	Elimina un InputSlot de la colección cuyo identificador es igual al valor pasado como parámetro.
Nombre:	InputSlot& getInputSlot(VarId id)
Descripción:	Retorna el slot de la colección cuyo identificador es igual al pasado como parámetro.
Nombre:	void retrieveInput( )

Descripción:	Recupera la información de cada Slot que se encuentra en la colección

Nombre: Updateable	
Descripción: Interfaz que define que un objeto sea actualizable.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual void update(Variable* variable) = 0

Nombre: Variable	
Descripción: Almacena el valor recogidos en el campo para una variable.	
Tipo de clase: Controladora	
Atributo	**Tipo**
id	VarId
value	VariableValue
timestamp	Timestamp
Para cada responsabilidad:	
Nombre:	const VarId getId ( )
Descripción:	Retorna el valor de atributo id
Nombre:	virtual VariableValue getValue() const
Descripción:	Retorna el valor del atributo value
Nombre:	virtual void setValue(VariableValue value)
Descripción:	Asigna un nuevo valor al atributo value.
Nombre:	Timestamp getTimestamp() const
Descripción:	Retorna el valor del atributo timestamp.

Nombre: VariableSet	
Descripción: Contiene la colección de los valores que se recogen en el campo para una variable.	
Tipo de clase: Controladora	
Atributo	**Tipo**
id	VarId
values	VariableValueCollection
timestamps	TimestampCollection
Para cada responsabilidad:	
Nombre:	VarId getId ( )
Descripción:	Retorna el valor del atributo id.
Nombre:	void setId (VarId id)
Descripción:	Asigna un nuevo valor al atributo id.
Nombre:	VariableValueCollection& getValues ( )
Descripción:	Retorna el valor del atributo values
Nombre:	TimestampCollection& getTimestamps( )
Descripción:	Retorna el valor del atributo timestamps.

Nombre: VariableIO	
Descripción: Interfaz que debe implementar todas las clases que manejen información de entrada y salida.	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	virtual const VariableCollection& getNextBatch( ) = 0

Conclusiones

En este capítulo se ha realizado la descripción de la mayor parte de los elementos implementados, se han seleccionado los subsistemas más significativos y sus principales clases. Los subsistemas *Utils*, *Texts*, *Events* y *GTK* no se han descrito porque reflejan poca complejidad, no dejan de ser importantes pero se han seleccionado solo los que facilitan en mayor medida la comprensión del código fuente.

Para la implementación definimos un estilo de códificación discutido profundamente entre los participantes en el desarrollo de la aplicación. Este estándar incluye elementos de buena práctica de programación, errores comunes que cometen los desarrolladores, manejo de memoria para evitar pérdidas de información, política de nombres y otros aspectos relevantes. La propuesta inicial la tomamos de un libro escrito por el creador del lenguaje C++: *Bjarne Stroustrup* y un grupo de destacados conocedores de la teoría de programación de sistemas, expertos que tomaron como referencia a varios proyectos con carácterísticas de riesgo y alta complejidad. Por último se definieron un grupo de aspectos que conformaron nuestro estándar de codificación.

Capítulo 3. Validación de la Solución

Introducción

Algunos filósofos de la programación han dicho que el ejercicio de su trabajo, de una manera consciente, lógica y cuidadosa no hace falta la realización de pruebas para garantizar el buen funcionamiento del código; esta es una afirmación cuestionable ya que el decursar de la poca historia que tiene el desarrollo de aplicaciones ha demostrado la necesidad de ellas, pues validan el correcto funcionamiento del trabajo realizado desde las etapas iniciales del cualquier proyecto.

En metodologías como Extreme Programing, su realización tiene un importante lugar en el desarrollo, al punto de estar vinculado directamente a la codificación. Con estas no se encontrarán todos los errores que tenga el producto, pero ayuda a garantizar que la mayor parte que conforma el código fuente (source code) cumpla su funcionalidad correctamente, no identifica errores de integración u optimización. Para realizarlas existen frameworks que ayudan en el proceso de automatización con funciones que mediante criterios verifiquen el correcto funcionamiento del código.

3.1 - Diseño de los test de unidades que permitan validar la solución propuesta

Para la realización de las pruebas se utilizan como recursos físicos: computadoras con un microprocesador Intel Pentium 4 con una velocidad del CPU de 3.2 Ghz, la memoria RAM que tienen es de 512 Mb y un disco duro con capacidad de 80 Gbytes.

Los recursos lógicos con los que realizamos las pruebas fueron: sistema operativo Debian GNU/Linux con núcleo 2.6.20, el IDE de desarrollo es el Eclipse 3.2 con el plug-in CDT 2.1 para

programar en C++, y el framework para automatizar las pruebas es el CxxTest, que se integra con el entorno de desarrollo y sirve para el lenguaje que utilizamos.

Todos los casos de pruebas que han sido realizados son del tipo Pruebas de Unidad y tienen como objetivo aislar el código fuente y validar el correcto funcionamiento de los métodos que se implementan. Se realizaron por la misma organización que tienen los subsistemas y las clases dentro del código. Nos hemos basado en el patrón de prueba “Rango de Parámetro”.

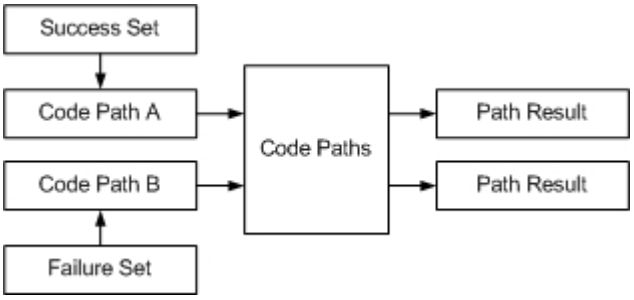

Figura 16: Patrón de Pruebas: Rango de Parámetro.

3.1.1 - Pruebas a las clases dentro del subsistema Base

Las clases Named, Titled, Object y Viewer son las que por su implementación existe la posibilidad de realizarle pruebas, las restantes clases son Interfaces, definiciones de tipo o simplemente no tienen recursos para realizarles pruebas ya que en la mayoría de los casos se reutilizan métodos de otras clases. Las clases probadas son Named, Object, Titled y Viewer.

Clase TestNamed

Prueba unitarias de la clase: **Named**

Casos de prueba del método: **void setName(Name name)**

Variables a considerar en el caso de prueba: **Name name**

Clase de equivalencia para la variable: **name**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Verificar que cambie el valor del atributo name.	Válida

Casos de prueba:

1- void testSetName( )

Tabla 1. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
	Verificar que se pueda cambiar el valor del atributo	Cadena pasada como parámetro.	El la misma cadena pasada en los datos de entrada	El la misma cadena pasada en los datos de entrada	

Clase TestObject

Prueba unitarias de la clase: **Object**

Casos de prueba del método: **Identifier getId()**

Variables a considerar en el caso de prueba: **Identifier id**

Clase de equivalencia para la variable: **id**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Variables id por debajo de los	Inválida

	valores acotados por el unsigned int	
2	Variables id entre los valores acotados por el unsigned int	Válida
3	Variables id por encima de los valores acotados por el unsigned int	Inválida

Casos de prueba:

1- void testGetId_01()

2- void testGetId_02()

3- void testGetId_03()

Tabla 2. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verifica que dado valores pertenecientes a la frontera de invalidez inferior se obtengan valores esperados	-1000	Error de asignación	El valor de la variable es 4294966296	No se obtuvo la salida esperada, ya que no esta validada esta partición.
2	Verifica que dado valores que se encuentren dentro de la frontera de validez se	1000	El la misma cadena pasada en los datos de entrada	El la misma cadena pasada en los datos de entrada	

	devuelvan los datos esperados				
3	Verifica que dado valores pertenecientes a la frontera de invalidez superior se obtengan valores esperados	88859857	Error de asignación	Error de asignación	

Clase TestTitled

Prueba unitarias de la clase: **Titled**

Casos de prueba del método: **void setTitle(Title title)**

Variables a considerar en el caso de prueba: **Title title**

Clase de equivalencia para la variable: **title**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Valores validos del atributo.	Válida

Casos de prueba:

1- void testSetTitle( )

Tabla 3. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**

1	Verifica que se pueda cambiar el valor del atributo	Cadena pasada como parámetro.	El la misma cadena pasada en los datos de entrada	El la misma cadena pasada en los datos de entrada	

Clase TestViewer

Prueba unitarias de la clase: **Viewer**

Casos de prueba del método: **void setIOManager( )**

Variables a considerar en el caso de prueba: **IOManager ioManager**

Clase de equivalencia para la variable: **ioManager**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Como la función a probar recibe una variable de tipo IOManager, se construye y se le pasa como parámetro.	Válida

Casos de prueba:

1- void testSetIOManager()

Tabla 4. Formato de pruebas unitarias del método.

Caso de	**Objetivo de la**	**Datos de**	**Salida**	**Salida**	**Observación**

prueba	prueba	entrada	esperada	obtenida	
1	Verifica que dado un valor nuevo antrado a la variable se retorne el esperado	IOManager* ioManager = new IOManager()	El la misma cadena pasada en los datos de entrada	El la misma cadena Tabla 1 pasada en los datos de entrada	

3.1.2 - Pruebas a las clases dentro del subsistema Cairo

La clase principal de este subsistema es CairoCanvas, la encargada de abstraer las funcionalidades que nos brinda la biblioteca gráfica Cairo, por esta razón solo se le hacen pruebas a los métodos que por su implementación permitan la realización de las mismas, ya que asumimos el buen funcionamiento de la biblioteca. Las clases probadas son CairoCanvas, SVGReader y PGReader.

Clase TestCairoCanvas

Pruebas unitarias de la clase: **CairoCanvas**

Casos de prueba del método: **getLineColor()**

Variables a considerar en el caso de prueba: **RGBAColor * color**

Clase de equivalencia para la variable: **color**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Como la función a probar recibe una variable tipo RGBAColor pasar valores	Válida

	válidos para el dominio de un objeto de tipo color como pueden se el valor de R, G, B, A los cuales siempre deben estar entre cero y 255.	
2	Valores no válidos para el dominio de un RGBAColor, al menos uno de los parámetros del color debe estar por debajo de cero.	Inválida
3	Valores no válidos para el dominio de un RGBAColor, al menos uno de los parámetros del color debe estar por encima de cero.	Inválida

Casos de prueba:

1- void testGetLineColor();

Tabla 5. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Que se le asigna correctamente el color pasado como parámetro a una línea determinada.	RGBAColor *color = new RGBAColor(255, 255, 255, 0) CairoCanvas canvas;	Color con los mismos valores pasados en los datos de entrada.	Color con los mismos valores pasados en los datos de entrada.	
2	Que se provoca alguna excepción al pasar un	RGBAColor *color = new RGBAColor(-1,	Excepción o algún tipo de notificación por parte de objeto	No se produce ninguna notificación ni	

	RGBAColor con valores no válidos.	255, 256, 0); CairoCanvas canvas;	cuando se le pasa como argumento un valor no válido.	excepción.	

Casos de prueba del método: **getFillColor()**

Variables a considerar en el caso de prueba: **RGBAColor * color**

Clase de equivalencia para la variable: **color**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Como la función a probar recibe una variable tipo RGBAColor pasar valores válidos para el dominio de un objeto de tipo color como pueden se el valor de R, G, B, A los cuales siempre deben estar entre 0 y 255.	Válida
2	Valores no válidos para el dominio de un RGBAColor, al menos uno de los parámetros del color debe estar por debajo de cero.	Inválida
3	Valores no válidos para el dominio de un RGBAColor, al menos uno de los parámetros del color debe estar por encima de cero.	Inválida

Casos de prueba:

1- testGetFillColor()

Tabla 6. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Que se le asigna correctamente el color pasado como parámetro.	RGBAColor *color = new RGBAColor(255, 255, 255, 0); CairoCanvas canvas;	Color con los mismos valores pasados en los datos de entrada.	Color con los mismos valores pasados en los datos de entrada.	
2	Que se provoca alguna excepción al pasar un RGBAColor con valores no válidos.	RGBAColor *color = new RGBAColor(-1, 255, 256, 0); CairoCanvas canvas;	Excepción o algún tipo de notificación por parte de objeto cuando se le pasa como argumento un valor no válido.	No se produce ninguna notificación ni excepción.	

Casos de prueba del método: **getBackgroundColor()**

Variables a considerar en el caso de prueba: **RGBAColor * color**

Clase de equivalencia para la variable: **color**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Como la función a probar recibe una variable tipo RGBAColor pasar valores válidos para el dominio de un objeto de tipo color como pueden se el valor de R, G, B, A los cuales siempre deben estar entre 0 y 255.	Válida
2	Valores no válidos para el dominio de un RGBAColor, al menos uno de los parámetros del color debe estar por debajo de cero.	Inválida
3	Valores no válidos para el dominio de un RGBAColor, al menos uno de los parámetros del color debe estar por encima de cero.	Inválida

Casos de prueba:

1- testGetBackgroundColor()

Tabla 7. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Que se le asigna correctamente el color pasado como parámetro.	RGBAColor *color = new RGBAColor(255, 255, 255, 0);	Color con los mismos valores pasados en los datos de entrada.	Color con los mismos valores pasados en los datos de entrada.	

		CairoCanvas canvas.			
2	Que se provoca alguna excepción al pasar un RGBAColor con valores no válidos.	RGBAColor *color = new RGBAColor(-1, 255, 256, 0); CairoCanvas canvas;	Excepción o algún tipo de notificación por parte de objeto cuando se le pasa como argumento un valor no válido.	No se produce ninguna notificación ni excepción.	

Casos de prueba del método: **getFontFace()**

Variables a considerar en el caso de prueba: **FontFace font**

Clase de equivalencia para la variable: **font**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Se le asignará a la variable un valor de tipo string y se pasará como parámetro.	Válida

Casos de prueba:

1- testGetFontFace()

Tabla 8. Formato de pruebas unitarias del método.

Caso de	**Objetivo de**	**Datos de**	**Salida**	**Salida**	**Observación**

prueba	la prueba	entrada	esperada	obtenida	
1	Que se le asigna correctamente el color pasado como parámetro.	Font = “ARIAL” CairoCanvas canvas;	Fuente con los mismos valores pasados en los datos de entrada.	Fuente con los mismos valores pasados en los datos de entrada.	

Casos de prueba del método: **getLinewidth()**

Variables a considerar en el caso de prueba: **WindowCoord coord**

Clase de equivalencia para la variable: **coord**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Un valor positivo para la variable coord de tipo WindowCoord.	Válida

Caso de prueba:

1- testGetLineWidth()

Tabla 9. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Que se le asigna y se retorna de forma correcta el valor de el ancho de la	WindowCoord Coord. = 10; CairoCanvas canvas;	El valor de retorno debe ser el mismo que se le asigna.	El valor retornado es 134922776	No se obtuvo la salida esperada.

	línea.				

Clase TestSVGReader

Prueba unitaria de la clase: **SVGReader**

Casos de prueba del método: **Load()**

Variables a considerar en el caso de prueba: **VectorImage image**

Clase de equivalencia para la variable: **image**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Todos los valores de píxel en NULL	Válida

Caso de prueba:

1- testLoad()

Tabla 10. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Comprobar que no se provocan excepciones	VectorImage image	No se debe provocar ningún tipo de error.	No se provoca error.	

Clase TestPNGReader

Prueba unitaria de la clase: **PNGReader**

Casos de prueba del método: **Load()**

Variables a considerar en el caso de prueba: **PixelSet pixel**

Clase de equivalencia para la variable: **pixel**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Todos los valores de píxel en NULL	Válida

Caso de prueba:

1- testLoad()

Tabla 11. Formato de pruebas unitarias del método**.**

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Comprobar que no se provocan excepciones	PixelSet pixel	No se debe provocar ningún tipo de error.	No se provoca error.	

3.1.3 - Pruebas a las clases dentro del subsistema Draw

En este se subsistema se encuentran clases que implementan funcionalidades heredadas de las interfaces y hacen una implementación particular para cada objeto gráfico.

Clase TestListenerCapableImpl

Prueba unitaria de la clase: **ListenerCapableImpl**

Casos de prueba del método: **setListener(RepaintListener** repaintListener)**

Variables a considerar en el caso de prueba: **RepaintListener** repaintListener**

Clase de equivalencia para la variable: **repaintListener**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Valor positivo para un punto.	Válida

Casos de prueba:

1- void TestsetListener()

Tabla 12. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verifica que se pueda cambiar el valor del atributo	RepaintListener ** rlistener = new RepaintListener ();	Un objeto igual al pasado como parámetro a la función.	Un objeto igual al pasado como parámetro a la función.	

3.1.4 - Pruebas a las clases dentro del subsistema Graphics

Las clases que se prueban en este subsistema sarán las que reutilizaran las clases para comportarse de una manera determinada, han sido probadas las clases Dephtable, FillableImpl, PositionableImpl, RGBAColor, Shape, Text, VectorImage y PixelSet.

Clase TestPositionableImpl

Pruebas unitarias de la clase: **PositionableImpl**

Casos de prueba del método: **setTopLeft(Point* topLeft)**

Variables a considerar en el caso de prueba: **Point* topLeft**

Clase de equivalencia para la variable: **topLeft**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Verificar que cambie el valor del atributo.	Válida

Casos de prueba:

1-void testsetTopLeft()

Tabla 13. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Verifica que se pueda cambiar el valor del atributo	Point* topLeft = new Point(0, 0)	Point* topLeft	Point* topLeft	

Casos de prueba del método: **setWidth(WindowCoord width)**

Variables a considerar en el caso de prueba: **WindowCoord width**

Clase de equivalencia para la variable: **width**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Valor de la variable dentro del rango valido de valores	Válida

Casos de prueba:

1- void testSetWidth()

Tabla 14. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Encontrar error al entrarle un nuevo valor a la variable.	35	35	35	

Casos de prueba del método: **setHeight(WindowCoord height)**

Variables a considerar en el caso de prueba: **WindowCoord height**

Clase de equivalencia para la variable: **height**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Valor de la variable dentro del	Válida

	rango valido de valores	

Casos de prueba:

1-void testsetHeight()

Tabla 15. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Encontar error al entrarle un nuevo valor a la variable.	300	300	300	

Clase TestDephableImpl

Pruebas unitarias de la clase: **DephableImpl**

Casos de prueba del método: **setDepht(Depth depth)**

Variables a considerar en el caso de prueba: **Depth depth**

Clase de equivalencia para la variable: **depth**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Verificar que cambie el valor del atributo dentro de los rangos	Válida

	válidos.	

Casos de prueba:

1-void testSetDepth()

Tabla 16. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verifica que se pueda cambiar el valor del atributo	20	20	20	

Clase TestFilliableImpl

Pruebas unitarias de la clase: **FilliableImpl**

Casos de prueba del método: **setFillColor(RGBAColor* color)**

Variables a considerar en el caso de prueba: **RGBAColor* color**

Clase de equivalencia para la variable: **color**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Verificar que cambie el valor del atributo dentro de los rangos válidos.	Válida

Casos de prueba:

1-void testSetFilliableColor()

Tabla 17. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verifica que se pueda cambiar el valor del atributo	RGBAColor *color = new RGBAColor(0, 0, 0, 0);	Color con los mismos datos de entrada.	Color con los mismos datos de entrada.	

Clase TestRGBAColor

Pruebas unitarias de la clase: **RGBAColor**

Casos de prueba para el métodos: **setRed(ColorDimension red)**

Variables a considerar en el caso de prueba: **ColorDimension red**

Clase de equivalencia para la variable: **red**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Rango de valores entre 0 y 255	Válida
2	Rango de valores por debajo de 0.	Clase no valida
3	Rango de valores por encima de 255.	Clase no valida

Casos de prueba:

1- void testSetRed_01()

2- void testSetRed_02()

3- void testSetRed_03()

Tabla 18. Formato de pruebas unitarias de los métodos.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Que se le asigne correctamente el valor del color pasado como parámetro.	100	100	100	
2	Que se le asigne a la variable el rango un valor dentro del rango	-50	0	206	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de datos
3	Verificar que el parámetro width este por debajo del rango.	300	255	44	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de datos

Casos de prueba de el métodos: **setGreen(ColorDimension green)**

Variables a considerar en el caso de prueba: **ColorDimension green**

Clase de equivalencia para la variable: **green**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Rango de valores entre 0 y 255	Válida
2	Rango de valores por debajo de 0.	Clase no válida
3	Rango de valores por encima de 255.	Clase no válida

Casos de prueba:

1- void testSetGreen_01()

2- void testSetGreen_02()

3- void testSetGreen_03()

Tabla 19. Formato de pruebas unitarias de los métodos.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verificar que el metodo funciona para un valor en el rango de 0-255	100	100	100	
2	Verificar que	-50	0	206	No se obtuvo

	el método funciona para un valor por debajo del rango.				la salida esperada ya que la clase no tiene validada la entrada de datos
3	Verificar que el parámetro width este por debajo del rango.	300	255	44	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de datos

Casos de prueba de el métodos: **setBlue(ColorDimension blue)**

Variables a considerar en el caso de prueba: **ColorDimension blue**

Clase de equivalencia para la variable: **blue**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Rango de valores entre 0 y 255	Válida
2	Rango de valores por debajo de 0.	Clase no valida
3	Rango de valores por encima de 255.	Clase no valida

Casos de prueba:

1- void testSetBlue_01()

2- void testSetBlue_02()

3- void testSetBlue_03()

Tabla 20. Formato de pruebas unitarias de los métodos.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verificar que el método funciona para un valor en el rango de 0-255	100	100	100	
2	Verificar que el metodo funciona para un valor por debajo del rango.	-50	0	206	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de datos
3	Verificar que el parámetro width este por debajo del rango.	300	255	44	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de datos

Casos de prueba de el métodos: **setAlpha(ColorDimension alpha)**

Variables a considerar en el caso de prueba: **ColorDimension alpha**

Clase de equivalencia para la variable: **alpha**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Rango de valores entre 0 y 255	Válida
2	Rango de valores por debajo de 0.	Clase no valida
3	Rango de valores por encima de 255.	Clase no valida

Casos de prueba:

1- void testSetAlpha_01()

2- void testSetAlpha_02()

3- void testSetAlpha_03()

Tabla 21. Formato de pruebas unitarias de los métodos.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Verificar que el método funciona para un valor en el rango de 0-255	100	100	100	
2	Verificar que	-50	0	206	No se obtuvo

	el método funciona para un valor por debajo del rango.				la salida esperada ya que la clase no tiene validada la entrada de datos
3	Verificar que el parámetro width este por debajo del rango.	300	255	44	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de datos

Casos de prueba para: **la sobrecarga del operador =**

Variables a considerar en el caso de prueba: **RGBAColor *color**

Clase de equivalencia para la variable: **color**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Valores correctos para una objeto de tipo RGBAColor.	Válida
2	Valores correctos para una objeto de tipo RGBAColor.	Clase no válida

Casos de prueba:

1- void testOperator_01()

1- void testOperator_02()

Tabla 22. Formato de pruebas unitarias del operador =.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verificar que el operador = redefinido funciona para un puntero de tipo RGBAColor instanciado.	RGBAColor* color = new RGBAColor(100,100,100,100);	Asignación correcta de la entrada a otra variable de tipo RGBAColor	Asignación correcta de la entrada a otra variable de tipo RGBAColor	
2	Verificar que el operador = redefinido no funciona para un puntero de tipo RGBAColor no instanciado.	RGBAColor* color = new RGBAColor(100,100,100,100);	Asignación correcta de la entrada a otra variable de tipo RGBAColor	Asignación correcta de la entrada a otra variable de tipo RGBAColor	

Clase TestShape

Pruebas unitarias de la clase: **Shape**

Casos de prueba del métodos: **setLineWidth (LineWidth width)**

Variables a considerar en el caso de prueba: **LineWidth width**

Clase de equivalencia para la variable: **width**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia

1	Rando de valores entre 0 y 65536.	Válida
2	Rando de valores por debajo 0.	Clase no valida
3	Rando de valores por encima de 65535.	Clase no valida

Casos de prueba:

1- void testSetLineWidth_01()

2- void testSetLineWidth_02()

3- void testSetLineWidth_03()

Tabla 23. Formato de pruebas unitarias de los métodos.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verifica que dado valores que se encuentren dentro de la frontera de validez se devuelvan los datos esperados	10	10	10	
2	Verificar que el método funciona para un valor por debajo del rango.	-10	1	4294967286	La clase no esta validada para la entrada de datos con valores negativos
3	Verificar que el parámetro width este por	65570	30	65570	La clase no esta validada para valores

	encima del rango.				por encima del rango esperado.

Casos de prueba de los métodos: **setLineColor(RGBAColor* color)**

Variables a considerar en el caso de prueba: **RGBAColor* color**

Clase de equivalencia para la variable: **color**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Valores correctos de color.	Válida
2	Valores incorrectos de color.	Clase no valida

Casos de prueba:

1- void testSetColor_01()

2- void testSetColor_02()

Tabla 24. Formato de pruebas unitarias de los métodos**.**

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Verificar que el método funciona para un parámetro de tipo	*color(13,2,5, 6)	*color(13,2,5, 6)	*color(13,2,5, 6)	

	RGBAColor.				
2	Verificar que el método funciona no para un parámetro de tipo RGBAColor.	&color(0,245, 51,6)	!&color(0,245, 51,6)	&color(0,245, 51,6)	La prueba falla porque este método es tan sencillo que para cualquier parámetro de tipo RGBAColor se ejecuta.

Clase TestText

Pruebas unitarias de la clase: **Text**

Casos de prueba de los métodos: **setText (String text)**

Variables a considerar en el caso de prueba: **String text**

Clase de equivalencia para la variable: **text**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Un texto de entrada.	Válida

Casos de prueba:

1- void testSetText_01()

Tabla 25. Formato de pruebas unitarias del método**.**

Caso de	**Objetivo de**	**Datos de**	**Salida**	**Salida**	**Observación**

prueba	la prueba	entrada	esperada	obtenida	
1	Verificar que los métodos funciona para un parámetro sencillo de tipo string	texto de prueba	texto de prueba	texto de prueba	

Clase TestLine

Pruebas unitarias de la clase: **Line**

Casos de prueba del método: **setInitialVertex (Point* initialVertex)**

Variables a considerar en el caso de prueba: **Point* initialVertex**

Clase de equivalencia para la variable: **inicialVertex**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Verificar que devuelve el valor correcto del atributo.	Válida

Casos de prueba:

1- void testSetInitialVertex()

Tabla 26. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verificar que devuelva el	(100, 200)	0	0	

	valor del atributo.				

Casos de prueba del método: **testSetFinalVertex(Point* finalVertex)**

Variables a considerar en el caso de prueba: **Point* finalVertex**

Clase de equivalencia para la variable: **final**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Verificar que devuelve el valor correcto de la variable.	Válida

Casos de prueba:

1- void TestGetFinalVertex()

Tabla 27. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Verificar que devuelva el valor del atributo	(20,20)	0	0	

Clase TestVectorImage

Pruebas unitarias de la clase: **VectorImage**

Casos de prueba del método: **setWidth (unsigned int width)**

Variables a considerar en el caso de prueba: **unsigned int width**

Clase de equivalencia para la variable: **width**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	El valor de la variable estará por encima de los valores validos	Inválida
2	El valor de la variable estará entre los valores permisibles de la variable.	Válida
3	El valor de la variable estará por encima de los valores permisibles de la variable.	Válida

Casos de prueba:

1- void testSetWidth()

Tabla 28. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verificar que al pasarle un número negativo, muestre un error ya que el unsigned int está acotado de 0 <= X <= 65535.	-34	Error de asignación	4294967262	Verificar por qué devuelve un número de 10 cifras, y validar la partición.

2	Verificar que al pasarle un valor dentro del rango devuelva el valor esperado.	3	3	3	
3	Verificar que al pasarle un valor por encima del rango muestre un error de asignación.	8546464	Error de asignación	9765464865	Esta partición no esta validada en el código de la aplicación

Casos de prueba del método: **testSetHeight (unsigned int height)**

Clase de equivalencia para la variable: **unsigned int height**

Clase de equivalencia para la variable: **height**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	El valor de la variable estará por encima de los valores validos	Inválida
2	El valor de la variable estará entre los valores permisibles de la variable.	Válida
3	El valor de la variable estará por encima de los valores permisibles de la variable.	Válida

Casos de prueba:

1- void testgetHeight( )

Tabla 28. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verificar que al pasarle un número negativo, muestre un error ya que el unsigned int está acotado de 0 <= X <= 65535.	-10	Error de asignación.	4294967262	Verificar por qué devuelve un número de 10 cifras, y validar la partición.
2	Verificar que al pasarle un valor dentro del rango devuelva el valor esperado.	300	300	300	
3	Verificar que al pasarle un valor por encima del rango muestre un error de asignación.	8546464	Error de asignación.	9765464865	Esta partición no esta validada en el código de la aplicación

Casos de prueba del método: **testSetData (SVGImage data)**

Variables a considerar en el caso de prueba: **SVGImage data**

Clase de equivalencia para la variable: **SVGImage data**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de

		equivalencia
1	Verificar el valor con el cual se inicializa la variable de tipo SVG.	Válida

Casos de prueba:

1- void testgetData_01( )

Tabla 29. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Verificar el valor con el cual se inicializa la variable de tipo SVG.	"data"	{00 00 00 00}	{00 00 00 00}	

Calse TestPixelSet

Pruebas unitarias de la clase: **PixelSet**

Casos de prueba del método: **setData(unsigned char* data)**

Variables a considerar en el caso de prueba: **unsigned char* data**

Clase de equivalencia para la variable: **data**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Verificar que cambie el valor del atributo.	Válida

Casos de prueba:

1-void testSetData()

Tabla 30. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verifica que se pueda cambiar el valor del atributo	3	3	3	

Casos de prueba del método: **setWidth(unsigned int width)**

Variables a considerar en el caso de prueba: **unsigned int width**

Clase de equivalencia para la variable: **width**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Verificar que cambie el valor del atributo.	Válida

Casos de prueba:

1- void testSetWidth()

Tabla 31. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verifica que se pueda cambiar el valor del atributo	200	200	200	

Casos de prueba del método: **setHeight(unsigned int height)**

Variables a considerar en el caso de prueba: **unsigned int height**

Clase de equivalencia para la variable: **height**

Clase de equivalencia	Clase de equivalencia	Clasificación de la clase de equivalencia
1	Verificar que cambie el valor del atributo.	Válida

Casos de prueba:

1-void testSetHeight()

Tabla 32. Formato de pruebas unitarias del método.

Caso de prueba	Objetivo de la prueba	Datos de entrada	Salida esperada	Salida obtenida	Observación
1	Verifica que se pueda cambiar	200	200	200	

	el valor del atributo				

Casos de prueba del método: **setFormat (Format format)**

Variables a considerar en el caso de prueba: **Format format**

Clase de equivalencia para la variable: **format**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Verificar que cambie el valor del atributo.	Válida

Casos de prueba:

1- void testSetFormat()

Tabla 33. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Verifica que se pueda cambiar el valor del atributo	Objeto de tipo RGBAColor	Objeto de tipo RGBAColor	Objeto de tipo RGBAColor	

3.1.5 - Pruebas a las clases dentro del subsistema IO

En este subsistema solo se han identificado dos clases a las que se pueden hacer pruebas: Variable y VariableSet.

Clase TestVariable

Pruebas unitarias de la clase: **Variable**

Casos de prueba del método: **setValue(VariableValue value)**

Variables a considerar en el caso de prueba: **VariableValue value**

Clase de equivalencia para la variable: **value**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Verificar que al pasarles números iguales de correcto.	Válida

Casos de prueba:

1- void testSetValue()

Tabla 34. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Verificar que al pasarles números iguales de correcto.	20	20	20	

Clase VariableSet

Pruebas unitarias de la clase: **VariableSet**

Casos de prueba del método: **setId(VarId id)**

Variables a considerar en el caso de prueba: **Var id**

Clase de equivalencia para la variable: **id**

Clase de equivalencia	**Clase de equivalencia**	**Clasificación de la clase de equivalencia**
1	Verificar que cambie el valor del atributo	Válida

Casos de prueba:

1- void testsetId()

Tabla 35. Formato de pruebas unitarias del método.

Caso de prueba	**Objetivo de la prueba**	**Datos de entrada**	**Salida esperada**	**Salida obtenida**	**Observación**
1	Verificar que cambie el valor del atributo.	Cadena de tipo String	Cadena de tipo String	Cadena de tipo String	

Conclusiones

Las pruebas unitarias arrojaron resultados de importancia considerable, en primer lugar ayudaron a identificar algunos errores que pasamos por alto en la codificación, principalmente en elementos que reutilizamos de experiencias anteriores; además pudieron validar el código fuente producido en la etapa de implementación.

Debido al tamaño del sistema y a su complejidad se identificaron un grupo de clases que podían ser “probadas” utilizando el patrón de pruebas Rango de Parámetro. Muchas de ellas constituían abstracciones que definían comportamientos y cualidades comunes con relación a otros conceptos derivados, por lo que no podían probarse en la definición abstracta y se hizo en las implementaciones concretas; otras clases utilizaban la implementación propuesta por elementos intermedios entre la definición y la implementación del concepto, en esos casos solo se realizaron pruebas a las clases “intermedias” sin repetirlas en las implementaciones concretas.

En general las pruebas unitarias ofrecieron valiosa información que ayudó a la corrección de los errores identificados y a implementar un producto con mayor calidad y en condiciones favorables para continuar su implementación y terminación.

Conclusiones

Para obtener el software resultante de esta investigación se transitó por varias etapas: primero la investigación y selección de las tecnologías a emplear, luego el análisis del diseño propuesto por los analistas, en tercer lugar la implementación de los subsistemas y por último se realizaron las pruebas de unidad al código fuente.

En sentido general se obtuvieron resultados satisfactorios que dieron cumplimiento a los objetivos propuestos:

- la HMI obtenida es capaz de representar gráficos en formato SVG y PNG, incluyendo animaciones básicas
- muestra valores históricos de las variables en gráficos de tendencia utilizando Cairo
- se implementó el soporte multilingue de la aplicación, en este momento tenemos diccionarios básicos en inglés y español
- se codificaron los conceptos definidos por el grupo de análisis y diseño de manera íntegra
- se diseñaron e implementaron las pruebas que validan la solución propuesta

Cumplidos estos objetivos enumeramos en la siguiente sección nuestras recomendaciones.

Recomendaciones

El producto que se obtuvo como resultado de esta investigación está en desarrollo, nuevas ideas y nuevas funcionalidades sugen cada día. Debemos recomendar dar seguimiento a la implementación para completar un sistema SCADA que impulse a Venezuela en la migración de sus sistemas al uso de software libre.

Recomendamos dirigir la continuación de la codificación en áreas como:

- serialización de los objetos que representan a los despliegues
- optimización de la representación gráfica de los gráficos vectoriales
- completamiento de una biblioteca de visualización de los gráficos de tendencia utilizando Cairo
- integración con el middleware y demás componentes del sistema SCADA
- completamiento del mecanismo de Entrada/Salida

Referencias Bibliográficas

[1] *Sistemas SCADA*. 2005. [2007]. Disponible en: http://www.automatas.org/redes/scadas.htm

[2] *Proyecto FreeScada*. 2005. [2007]. Disponible en: http://www.freescada.com/freescada.php

[3] *Proyecto Visual*. 2004. [2007]. Disponible en: http://visual.sourceforge.net/new/index.php

[4] *Proyecto QScada*. 2006. [2007]. Disponible en: http://qscada.sourceforge.net/

[5] *GIMP ToolKit*, [En Linea]. Fundación GTK, 2004. [2007]. Disponible en: http://www.gtk.org/

[6] *Documentación Oficial de Gtkmm*, [En Linea]. Fundación GTK, 2005. [2007]. Disponible en: http://www.gtkmm.org/gtkmm2/docs/

[7] *Elementos técnicos del producto Qt*, [En Linea]. Trolltech, 2006. [2007]. Disponible en: http://www.trolltech.com/products/qt/features/index

[8] *Licencia QPL*, [En Linea]. Trolltech, 1999. [2007]. Disponible en: http://www.trolltech.com/products/qt/licenses/licensing/qpl

[9] *Biblioteca gráfica Cairo*, [En Linea]. 2005. [2007]. Disponible en: http://www.cairographics.org

[10] JACKSON, D. and C. LILLEY. *About SVG, 2d Graphics in XML*, [En Linea]. 2004. [2005]. Disponible en: http://www.w3.org/Graphics/SVG/About

[11] *SVG Concepts*, [En Linea]. W3C, 2005. [2007]. Disponible en: http://www.w3.org/TR/SVG/concepts.html

[12] MAS, J. M. *SVG: Presente y futuro de los gráficos vectoriales para la web*, [En Linea]. 2005. [2007]. Disponible en: http://www.programacion.net/articulo/joa_svg1/

[13] *Documentación oficial de Gettext*, [En Linea]. GNU Project, 2003. [2007]. Disponible en: http://www.gnu.org/software/gettext/

[14] *Documentación de i18n*. [En Linea]. Proyecto Debian GNU/Linux, 2004. [2007]. Disponible en: http://www.debian.org/doc/manuals/intro-i18n/

[15] RAMEY, R. *Boost Libraries*. *Boost::Serialization*, [En Linea]. Boost, 2002-2004. Disponible en: http://www.boost.org/libs/serialization/doc/index.html

[16] SUGAR, D. and D. SILVERSTONE. *Common C++ Libraries*, [En Linea]. GNU, 2004. Disponible en: http://www.gnu.org/software/commoncpp/

[17] NGUYEN, B. *Linux Dictionary. PLplot*, [En Linea]. Debian Project, 2004. [2007]. Disponible en: http://linux.about.com/cs/linux101/g/plplot.htm

[18] JACOBSON, I.; BOOCH, G.; Rumbaugh, J. *El Proceso Unificado de Desarrollo de Software*. Addison Wesley, 1999. 4 p.

[19] JACOBSON, I.; BOOCH, G.; Rumbaugh, J. *El Lenguaje Unificado de Modelado. Manual de Referencia*. Addison Wesley, 2000. 3 p.

Otras bibliografías consultadas

The GNU Project, [En Linea]. GNU Project, 2003. [2007]. Disponible en: http://www.gnu.org

Fundación del Software Libre, [En Linea]. FSF, 1999. [2007]. Disponible en: http://www.fsf.org/

Productos de la compañía Trolltech, [En Linea]. Trolltech, 2006. [2007]. Disponible en: http://www.trolltech.com/

Glosario de Términos

SCADA: acrónimo de Supervisory Control and Data Acquisition. Sistema que se encarga de capturar y manipular la información referente a un proceso automatizado, esta información es utilizada para la realización análisis de indicadores y en la retroalimentación sobre algún operador.

Unidad de Terminal Remota (RTU): dispositivo instalado en lugares remotos que recolectan la información utilizando equipos medidores en el campo donde se opera.

Unidad Maestra (MTU): dispositivo (generalmente un ordenador) que manipula la información que proviene del campo a través de las RTU, esta asociado a la HMI y sirve de enlace entre esta y las RTUs.

Interfaz Hombre-Máquina (HMI): aplicación de software (casi siempre una interfaz gráfica de usuario) que permite la interacción entre el hombre y la máquina de manera amigable y sencilla.

Controlador Lógico Programable (PLC): dispositivo electrónico muy usado en la esfera de la automatización, tienen la capacidad de interactuar con otros dispositivos de campo, realizar operaciones aritméticas, interpretar señales analógicas y digitales, ente otras.

Serialización: proceso de codificación de un objeto (programación orientada a objetos) en un medio de almacenamiento (archivo o un buffer de memoria) como una serie de bytes o en un formato humanamente más legible como XML.

Widget: componente gráfico o visual con el que interactúa el usuario en las aplicaciones de software.