Universidad de las Ciencias Informáticas

Facultad 5

Título:

"Implementación del módulo de diseño de reportes para el SCADA Guardián del Alba"

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Ernesto Leyva Barrero

Tutor(es): Ing. Bernardo Zaragoza Hijuelos

Ing. Raudi Agdel Bacallao Sánchez

Ciudad de la Habana, 2009

Datos de Contacto

Tutor(es):

Ing. Bernardo Zaragoza Hijuelos

Graduado de Ingeniero en Ciencias Informáticas y profesor de la Universidad de Ciencias Informáticas (UCI), con 3 años en el desarrollo de software.

Ing. Raudi Agdel Bacallao Sánchez

Graduado de Ingeniero en Ciencias Informáticas y profesor de la Universidad de Ciencias Informáticas (UCI), con 3 años en el desarrollo de software.

Agradecimientos

A todos mis compañeros de estudio y trabajo del proyecto SCADA, por su incondicional ayuda en todos estos años y demostrar que somos un gran equipo.

A mis tutores que siempre han estado guiándome y formándome como un mejor profesional y sobre todo son mis amigos.

A mi familia toda, en especial a mi madre y mis hermanos, por confiar en mí y darme siempre la fuerza y el valor para seguir adelante, aun cuando las condiciones sean adversas.

A mi querida Lisbett, por su comprensión y ayuda en la realización de este trabajo, dedicando largas horas de revisión.

A mis amigos, que son lo más grande que pueda tener un ser humano.

A mi familia completa, sin excepción, les dedico mi trabajo de diploma, y más que eso, les dedico mi vida, la que hasta hoy me han enseñado a vivir, y la que viviré de aquí en lo adelante pues todo lo que soy se los debo a ustedes y especial a mis padres Teresa y Jorge. Papá donde quiera que estés esto es el fruto de lo que con tanto amor formaste junto a mamá.

A mis amigos, todos, que me han ayudado a ser mejor cada día y estoy muy agradecido de tenerlos como amigos y más que eso, como hermanos.

A mi Liss, y familia, por ayudarme tanto y demostrarme que el valor de las personas nace con ellas.

RESUMEN

Producto del convenio entre Cuba y la República Bolivariana de Venezuela y bajo la rectoría de las empresas ALBET-PDVSA surge un proyecto de software nombrado SCADA Nacional, el cual por sus siglas en inglés (Supervisory Control and Data Adquisition) se trata de un sistema de supervisión, control y adquisición de datos, cuyo principal objetivo es lograr primeramente la independencia tecnológica en esta importante industria y a la vez en el plano técnico, automatizar el proceso productivo de Petróleos de Venezuela. Para el sistema es necesario tener un subsistema que se encargue del diseño y generación de los informes relacionados con los datos históricos almacenados en las bases de datos de este tipo. En aras de dar solución a este problema se realiza este trabajo de diploma cuyo principal objetivo es implementar un diseñador de reportes sobre plataforma de software libre, capaz de interactuar con los correspondientes módulos del SCADA y que permita la configuración de los informes deseados.

Para llevar a cabo este objetivo, se realiza un estudio minucioso de los principales diseñadores de informes, esencialmente los que están desarrollados sobre software libre. Además se efectúa un análisis detallado de las principales tecnologías de desarrollo posibles a utilizar y se hace una propuesta de las más importantes. Se presentan además las pruebas realizadas al código escrito con el objetivo de que el software cumpla con los principios de calidad pertinentes y tenga un buen estado desde el punto de vista funcional.

PALABRAS CLAVES

SCADA, ingeniería, requisitos, software, informe, reporte, diseño, clases.

Tabla de contenidos

DATOS DEL CONTACTO.....II

AGRADECIMIENTOS.....III

AGRADECIMIENTOS.....III

DEDICATORIA.....IV

RESUMEN.....V

RESUMEN V

ÍNDICE DE FIGURAS.....XI

ÍNDICE DE FIGURAS.....XI

INTRODUCCIÓN.....1

FUNDAMENTACIÓN DEL TEMA.....5

1.1 Generalidades sobre los generadores de Reportes.....5

1.2 Diseño de informes.....7

1.3 Los reportes en los SCADAs.....7

1.4 Definición de SCADA.....8

1.5 Los SCADAs en la actualidad.....9

1.6 Generalidades de los SCADAs.....9

1.7 Diseñadores de reportes sobre plataforma de Software Libre.....10

1.7.1 NCReport.....11

1.7.2 Report Manager 11

1.7.3 Kugar.....12

1.7.4 Rekall 15

.....16

1.7.5 Jasper Reports.....16

1.7.6 Data Vision *17*

1.8 PRINCIPALES TENDENCIAS Y TECNOLOGÍAS 18

1.8.1 Software libre *19*

1.8.2 Metodologías de Desarrollo de Software *19*

1.8.2.1 Metodologías “Pesadas” 19

1.8.2.2 Metodologías “Ágiles” 20

1.8.2.3 Programación Extrema (XP): 20

1.8.2.4 Funcionalidad FDD 21

1.8.3 UML como lenguaje de modelado *22*

1.8.4 C++ *22*

1.8.5 Arquitectura de tres capas. *23*

1.8.5.1 Capa de presentación 24

1.8.5.2 Capa de negocio 24

1.8.5.3 Capa de datos 25

1.8.6 Patrón arquitectónico Modelo Vista Controlador (MVC) *25*

1.8.6.1 Modelo 25

Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos 25

1.8.6.2 Vista 25

1.8.6.3 Controlador 25

1.8.7 Biblioteca Qt *26*

1.8.8 Biblioteca Gráfica GTK *26*

1.8.9 Cairo *26*

1.8.10 Biblioteca Boost. *27*

1.8.11 IDEs *27*

1.8.11.1 KDevelop 28

1.8.11.2 Anjuta 29

1.8.11.3 Eclipse 29

1.8.11.4 Glade 30

1.8.11.5 CxxTest 30

1.8.11.6 Biblioteca GSL 31

1.9 TENDENCIAS PARA EL DESARROLLO 31

1.10 PROPUESTA 31

1.11 CONCLUSIONES 32

CONSTRUCCIÓN DE LA SOLUCIÓN PROPUESTA....33

1.12 Interacción del módulo Diseñador de Reportes....33

1.13 Patrones....34

1.13.1 Arquitectura Propuesta....34

1.13.1.1 Presentación....35

1.13.1.2 Modelo....35

1.14 Funcionamiento del diseñador y relación entre los dominios....35

1.14.1 Crear Reporte....36

1.14.1.1 Descripción de la Principales Clases....38

1.14.2 Propiedades generales del diseño....42

1.14.3 Propiedades de las páginas....43

1.14.4 Secciones....46

1.14.5 Descripción de las clases....61

1.15 Funcionalidades y propiedades de los componentes....65

....65

1.15.1 Componentes visuales para un diseño....65

1.15.1.1 Rectángulo y Círculo....68

1.15.1.2 Etiqueta, Fecha y Número de Página 70

1.15.1.3 Imagen....73

1.15.1.4 Línea....74

1.15.2 Funcionalidades en el diseño....76

1.15.2.1 Copiar - Pegar....76

1.15.2.2 Eliminar....77

1.15.2.3 Duplicar....77

1.15.2.4 Traer al frente y enviar al fondo....78

1.16 Conclusiones....79

VALIDACIÓN DE LA SOLUCIÓN....80

3.1 Prueba de Unidad....80

3.2 Diseño de las pruebas de unidad....82

3.3 Pruebas a las clases dentro del escenario "Crear Reporte" 82

3.3.1 Clase TestTreeNodeReport....82

3.3.1.1 Casos de Prueba:....83

3.3.1.2 Casos de Prueba:....84
3.3.2 Clase TestShowProperties....84
3.3.2.1 3.3.2.1 Casos de Prueba:....84
3.3.2.2 Casos de Prueba:....86
3.3.2.3 Casos de Prueba:....86
3.3.3 Clase TestPropertiesInspectorReport....87
3.3.3.1 Casos de Prueba:....87
3.3.3.2 Casos de Prueba:....88
3.3.4 Clase TestPropertiesInspectorReportPage....88
3.3.4.1 Casos de Prueba:....89
3.3.5 Clase TestPropertiesInspectorReportSection....89
3.3.5.1 Casos de Prueba:....90
3.3.5.2 Casos de Prueba:....90
3.4 Pruebas a las clases dentro del escenario "Insertar Componente"....91
3.4.1 Clase TestAddObject....91
3.4.1.1 Casos de Prueba:....92
3.4.1.2 Casos de Prueba:....93
3.4.1.3 Casos de Prueba:....93
3.4.1.4 Casos de Prueba:....94
3.4.2 Clase TestSection....95
3.4.2.1 Casos de Prueba:....96
3.4.2.2 Casos de Prueba:....96
3.4.3 Clase TestGraphic....97
3.4.3.1 Casos de Prueba:....98
3.4.3.2 Casos de Prueba:....99
3.4.3.3 Casos de Prueba:....100
3.4.3.4 Casos de Prueba:....101
3.4.4 Clase TestSelectTool....101
3.4.4.1 Casos de Prueba:....102
3.4.4.2 Casos de Prueba:....103
3.4.4.3 Casos de Prueba:....103
3.4.5 Clase Test ReportEditorView....104
3.4.5.1 Casos de Prueba:....104
3.4.5.2 Casos de Prueba:....105

3.4.5.3 Casos de Prueba:....106
3.4.5.4 Casos de Prueba:....107
3.5.1 Clase TestPropertiesInspectorReportGraphicObject....107
3.5.1.1 Casos de Prueba:....108
3.4.5.5 Casos de Prueba:....108
3.4.5.6 Casos de Prueba:....109
3.5.1.2 Casos de Prueba:....110
3.6 Conclusiones Parciales....110

CONCLUSIONES....112

RECOMENDACIONES....113

REFERENCIAS BIBLIOGRÁFICAS....114

BIBLIOGRAFÍA....114

Glosario de términos....115

Índice de Figuras

INTRODUCCIÓN

Con el desarrollo de la ciencia y la técnica, así como el exponencial auge de la automatización de los procesos industriales, las producciones de las grandes industrias han alcanzado niveles insuperables en comparación con momentos del pasado donde estas ramas no estaban tan desarrolladas como en la actualidad. La competencia del elevado número de empresas las ha llevado a usar la automática en sus instalaciones, aparejada al uso de la informática para controlar desde un ordenador los distintos procesos. Todo esto trae consigo un menor gasto de producción y en muchos casos se ejecutan operaciones en la producción imposibles de realizar por la mano del hombre, como son medir la temperatura de una caldera, el nivel de líquido de un tanque, entre otros.

Lo antes mencionado nos da una idea de la gran evolución que han tenido los sistemas digitales de control, supervisión y adquisición de datos (SCADAs) logrando ser el núcleo de los sistemas software de automatización de procesos e imponiéndose como pilar estratégico y fundamental para las grandes empresas industrializadas.

Los SCADAs son sistemas de software diseñados para funcionar sobre ordenadores y su principal objetivo es el control de las producciones, esto se logra interactuando directamente con los dispositivos automáticos, nombrados dispositivos de campo (controladores autónomos y autómatas programables), a través de redes LAN o buses especiales, y visualizando en tiempo real el comportamiento de dichos dispositivos en las consolas de operación para un fácil control por el personal laboral el cual debe tener los distintos niveles de privilegios para la explotación de los SCADAs, haciendo a estos sistemas más seguros y confiables.

Desde hace ya algunos años nuestro país viene haciendo innumerables esfuerzos por potenciar la automatización de las diferentes industrias, buscando mayor productividad en función de la calidad, ahorro de recursos y mejoras de nuestros procesos productivos.

En la Universidad de las Ciencias Informáticas se desarrollan numerosos proyectos productivos sobre automatización de empresas e instituciones tanto del país como en el exterior. Tal es el caso del “Guardián del Alba”, desarrollado por la Facultad 5 para la puesta en funcionamiento en las instalaciones

productivas de la empresa venezolana PDVSA (Petróleos de Venezuela). El objetivo fundamental del proyecto, es desarrollar un sistema de supervisión, control y adquisición de datos (SCADA), sobre la plataforma GNU/Linux bajo los preceptos de software libre y código abierto.

Debido a la alta dependencia tecnológica inherente a los productos existentes en el mercado, el cliente tomó la decisión de desarrollar un SCADA para la puesta en funcionamiento en sus instalaciones, que pudiera utilizarse también en otras instalaciones industriales y cuyo desarrollo estaría sujeto a los preceptos de software libre.

Existen argumentos técnicos sólidos que sustentan lo problemático de la situación presentada. El sistema OASyS, actualmente el SCADA más grande de PDVSA, es una inmensa infraestructura sobre la cual se encuentran configuradas el 70% de las señales automatizadas provenientes de los dispositivos de campo (PLC´s, RTU´s), y sobre el cual se maneja una producción aproximada de 1,1 MMBPD (Millones de barriles promedio por día), distribuida en 9 Unidades de Explotación, 8 Patios de Tanques y 2 Unidades de Servicio. La plataforma computacional que soporta el SCADA OASyS se encuentra actualmente en niveles críticos de capacidad y vida útil, existen problemas serios a nivel de hardware que afectan la disponibilidad y confiabilidad del sistema y que no permiten afrontar el crecimiento y la demanda de nuevos requerimientos para el crecimiento que necesita PDVSA Occidente, PDVSA y Venezuela.

Son muchas las funcionalidades que tienen los SCADAs y entre las básicas está la de proporcionar toda la información que se genera en el proceso productivo a diversos usuarios relacionados directamente con el control y supervisión de dichos procesos, así como a otros pertenecientes a niveles superiores dentro o fuera de la empresa como el control de calidad, supervisión y mantenimiento. Los valores instantáneos de las variables de proceso, los eventos, las alarmas y la configuración del sistema, entre otras, se registran en las bases de datos históricas del SCADA, y de toda esta data, solo una parte de ella resulta de interés para usuarios como los directivos y otros. Ejemplo: estadísticas del proceso, detección de fugas, deterioro de indicadores productivos y gráficas de comportamiento de diferentes procesos. Es por esto que se necesita que todo sistema SCADA tenga mecanismos para la generación de informes.

Los informes deben ser capaces de consolidar la información adquirida para entregarla en un formato determinado, de tal manera que sea útil al usuario al cual va dirigida. Es una realidad que la generación

de informes constituye una actividad crítica en muchas industrias, ya que se dedican largas jornadas delante de la pantalla para organizar la información, o hacer agrupaciones y conseguir que todo salga correctamente impreso. De esta afirmación no escapa la realidad presente hoy en los sistemas instaurados en PDVSA. De aquí la necesidad de que en el proyecto "Guardián del Alba" se desarrolle un módulo para la configuración y generación de reportes para darle solución al problema de la inexistencia de una herramienta capaz de diseñar y generar reportes, que sea lo suficientemente flexible como para extraer y procesar de forma automática la información almacenada en las bases de datos históricas del SCADA.

Si le añadimos las deficiencias en el proceso de elaboración y generación de reportes que contienen actualmente los SCADAs usados por PDVSA al involucrar varias herramientas propietarias, se hace muy difícil el proceso de configuración y parametrización de los informes ya que no existe un diseñador que permita definir plantillas para los reportes a generar ni un generador que emplee dichas plantillas.

Para dar solución a esta problemática se tiene que responder la presente interrogante como **Problema Científico**: ¿Cómo realizar la implementación de un diseñador de informes sobre plataforma GNU/Linux?

El **objeto de estudio** en la presente investigación es la configuración o diseño de informes bajo preceptos de software libre, dentro del cual se define como **campo de acción** la elaboración y configuración de informes basada en software libre, específicamente aplicada al sistema "Guardián del Alba"**.**

El **objetivo** del trabajo es implementar un módulo para la elaboración y configuración de reportes utilizando soluciones y herramientas libres que, al interactuar con el resto de los módulos del SCADA, permita configurar los reportes necesarios.

El desarrollo de este subsistema y su puesta en explotación de conjunto con el sistema SCADA permitirá que se minimice el tiempo de trabajo para llevar a cabo cualquier operación, así como evitar la pérdida de datos. Además permitirá la configuración personalizada de cualquier informe, los cuales una vez generados en los formatos pertinentes serán de vital importancia en la toma de decisiones.

De acuerdo al objetivo expuesto se definen los siguientes **Objetivos Específicos** para lograr un mayor

nivel de detalles en la investigación:

- Evaluar información sobre diseñadores de informes existente sobre plataformas de software libre.
- Estudiar las distintas tecnologías de desarrollo sobre software libre y en especial las principales bibliotecas para el desarrollo de aplicaciones.
- Implementar el módulo de configuración o diseño de reportes siguiendo la arquitectura propuesta y lograr la reutilización de los distintos componentes.
- Evaluar y analizar herramientas para la ejecución de pruebas de unidad.
- Realizar pruebas de unidad que permitan definir la robustez del sistema.

Esperando obtener como **resultados** los enunciados a continuación:

- Un módulo para el diseño o configuración de reportes que reúna las características y funcionalidades necesarias para su correcto uso y funcionamiento en un SCADA.
- Un software que cumpla con los objetivos para los cuales fue implementado, efectuando el diseño y ejecución de las pruebas necesarias.

El presente documento está estructurado en tres capítulos:

En el **Capítulo 1** se realiza un esbozo teórico centrado en temáticas como generadores de informes, sistemas SCADAs y las principales tendencias y tecnologías actuales que sirven de apoyo a todo el trabajo desarrollado.

En el **Capítulo 2** se analiza la arquitectura propuesta y se explican las principales funcionalidades.

El **Capítulo 3** se dedica a las pruebas de unidad realizadas para garantizar la integridad de la aplicación desarrollada.

CAPITULO

I

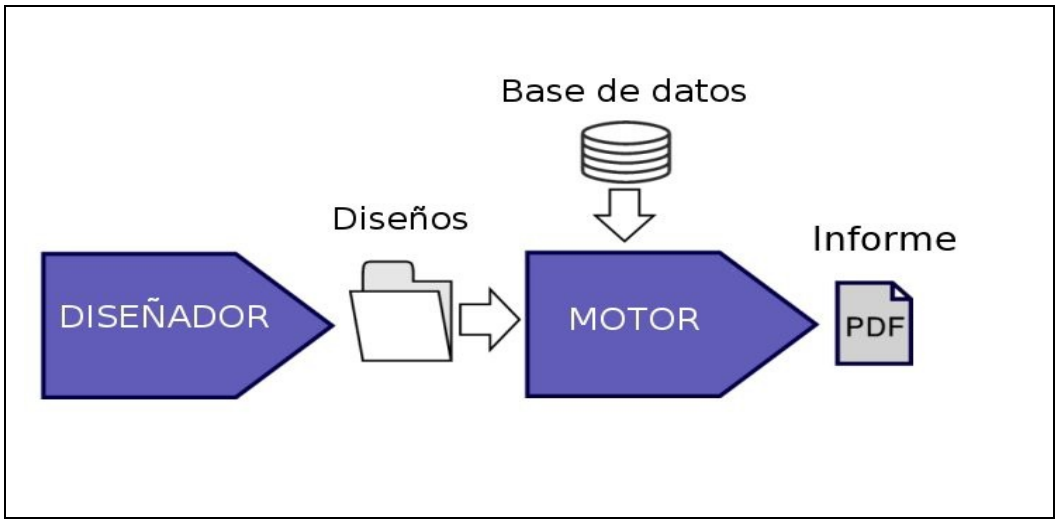
FUNDAMENTACIÓN DEL TEMA

En este capítulo se hace alusión a algunos conceptos y características generales de los generadores de reportes y la presencia en alguno de ellos de diseñadores de informes. Además se mencionan vínculos de estos sistemas con los SCADAs, de los cuales se presentan generalidades, se exponen algunos sistemas de visualización, y en especial, diseñadores de reportes con el objetivo de abordar las principales tecnologías que usan, además de varias bibliotecas gráficas de las más importantes usadas en la plataforma de software libre.

1.1 Generalidades sobre los generadores de Reportes.

Los generadores de reportes se componen fundamentalmente de dos elementos básicos, un diseñador o editor de informes y un motor de generación.

Fig. 1: Estructura general de un generador de reportes. Esquema de funcionamiento.



Como muestra la figura, las principales tareas que debe realizar un generador de reportes son las siguientes:

- Configurar el diseño y la apariencia que van a tener los reportes.
- Obtener los datos necesarios y elaborar el informe final de forma ordenada con la apariencia previamente configurada o diseñada.

Es por ello, la importancia de una herramienta capaz de llevar a cabo el diseño o edición de los reportes. La misma debe permitir que la configuración deseada se exporte en plantillas con un formato determinado, de manera tal que el generador posteriormente las utilice para volcar la data proveniente de las bases de datos y lograr generar el reporte deseado con la apariencia definida en la edición y obtenida de la plantilla.

Se conoce que varios generadores de informes presentan solo el componente de generación o motor generador, y la manera que tienen para diferenciar la apariencia visual de un reporte a otro es modificando las plantillas, a las cuales les definen una estructura y formato por defecto. Esta técnica es muy usada pero es un poco engorrosa puesto que no se ve el diseño de cómo deseamos sea nuestro reporte final.

1.2 Diseño de informes

El diseño de un reporte o informe no es más que una plantilla que modela y encierra las características con las cuales el usuario quiere que se genere el reporte deseado. De esta manera una vez que se insertan todos los componentes deseados en tiempo de diseño, en las distintas secciones, persistirá en la plantilla estas características y componentes para que la hora de generación el reporte se vea como mismo se diseñó pero con los datos volcados en su interior.

Es válido aclarar que los diseños de reportes que debemos permitir crear deben tener la apariencia adecuada para un sistema SCADA por lo cual será necesario definir y especificar todo a una serie de propiedades y mecanismos que permitan crear plantillas que puedan ser usadas desde el punto de vista funcional en un SCADA.

1.3 Los reportes en los SCADAs.

En la actualidad, se hace cada vez más necesaria la presencia en los SCADAs de subsistemas que permitan la generación de informes referente a la adquisición y registro de datos, generación de alarmas, entre otros, siendo esto la parte fundamental para la toma de decisiones. Es una realidad de nuestros días la existencia de una gran cantidad de proyectos enfrascados en el desarrollo de generadores de reportes, aunque ninguno de los encontrados se enfoca directamente en la generación de reportes para un SCADA ya que no incorporan en su diseño de aplicación los niveles de seguridad necesarios para manejar la integridad de la información obtenida, ni cuentan con mecanismos para la generación automática programada previamente por un operador. Es válido destacar que los generadores de reportes son, de los componentes de un SCADA, aplicaciones informáticas de las que más comúnmente encontramos en nuestros días en las cuales se expone una interfaz que visualiza la información obtenida de una base de datos, con la particularidad de que estos se vuelcan en la apariencia que fue previamente diseñada.

Resulta muy necesario realizar un estudio minucioso de muchos de estos proyectos con el objetivo de extraer formas de solución, mecanismos y resultados de los mismos y que puedan ser usados en la realización del presente trabajo. Además, el estudio se enmarca en la búsqueda de los generadores

realizados en la plataforma de software libre, aunque también se estudian algunos desarrollados para plataforma propietaria.

No se debe dejar de mencionar, por lo importante que resulta, que los reportes en los primeros SCADA se hacían vigilando las señales en los campos de forma periódica y así se llevaba la medida de las condiciones en que se encontraba el mismo, no se contaba con un motor de generación de informes ni un editor de reportes que le permitieran al operador hacer sus propios diseños.

Para la realización y visualización de los reportes, es necesario tener en cuenta que los mismos se alimentan de la información almacenada históricamente en las distintas bases de datos. Es por ello que la generación de informes es el principal punto para la toma de decisiones sobre la producción por parte del personal encargado.

De esta forma se hace necesario juntar las ideas y crear las bases para la implementación de dos módulos: uno para la edición y el otro para la visualización de reportes, que en conjunto formen el sistema generador de reportes para un SCADA. Según las necesidades del cliente el sistema debe ser lo más sencillo y amigable posible y a su vez de una manera robusta y fiable debe permitir la visualización del reporte final y una vez diseñado exportarlo a los distintos formatos definidos, como son pdf, csv, html, entre otros, además de seguir los preceptos de desarrollo utilizando herramientas de software libre.

Por todo lo antes descrito se hace necesario asumir la tarea de realizar el diseñador de reportes, el cual es de gran importancia para que los usuarios que usen el sistema SCADA puedan configurarse sus reportes a conveniencia.

1.4 Definición de SCADA

Como las siglas lo indican SCADA es el acrónimo en inglés de "Supervisory Control And Data Adquisition", es decir: sistema de control, supervisión y adquisición de datos. Generalmente se define: “como una forma de control remoto con disponibilidad para el control selectivo de las unidades remotas” (IEEE 1982). Otro concepto o definición de SCADA es la de ser un “aplicación software especialmente diseñada para funcionar sobre computadoras de control de producción, con acceso a la planta mediante

comunicación digital. Dicha comunicación se realiza con los reguladores locales básicos, e interfaces gráficas de alto nivel" (ARINA OYAGA Marzo de 2000).

1.5 Los SCADAs en la actualidad

Los SCADAs son sistemas diseñados para ser implementados o usados en numerosas industrias, contando con módulos de software genéricos disponibles para proporcionar las capacidades requeridas por las distintas entidades. La mayoría de los SCADAs que se instalan en la actualidad, se han convertido en una parte integral en la gestión de la información corporativa. Estos sistemas ya no solamente son vistos por la gerencia como herramientas operacionales, sino, además, como un recurso importante de información y esto se pone de manifiesto precisamente porque no solo cuentan con la interfaz visual que ve el operador, sino que tienen recursos que se exportan a otros sectores como son la plataforma web y la tecnología móvil, permitiendo a usuarios externos, como son los directivos y otros, ver lo que sucede en la producción controlada por el SCADA, así como obtener distintos reportes relacionados con datos importantes. Además de lo antes mencionado los actuales SCADAs alcanzan un nivel aceptable de tolerancia de fallas y cuentan con ordenadores redundantes operando en paralelo con el mismo SCADA en el centro primario del control y un sistema de reserva del mismo situado en un área geográficamente distante. Esta arquitectura permite la transferencia automática de la responsabilidad del control de cualquier ordenador que pueda tener cualquier tipo de colisión o falla por cualquier razón, a una computadora de reserva en línea permitiendo que no se detenga el servicio bajo ningún concepto.

1.6 Generalidades de los SCADAs

De forma general los SCADAs deben cumplir ciertos y determinados requerimientos para su fácil instalación, configuración, mantenimiento y puesta en funcionamiento. Podemos decir que deben ser sistemas de arquitectura abierta, capaces de crecer o adaptarse según las necesidades cambiantes de la empresa. También deben comunicarse con toda facilidad y de forma transparente al usuario con el equipo de planta y con el resto de la empresa (redes locales y de gestión). Además, deben ser programas sencillos de instalar, sin excesivas exigencias de hardware, y fáciles de utilizar, permitiendo interfaces de usuario amigables.

Es un aspecto principal en los SCADAs, el tratamiento y manejo de los datos adquiridos, así como su almacenamiento.

A continuación se expone un esquema de un SCADA:

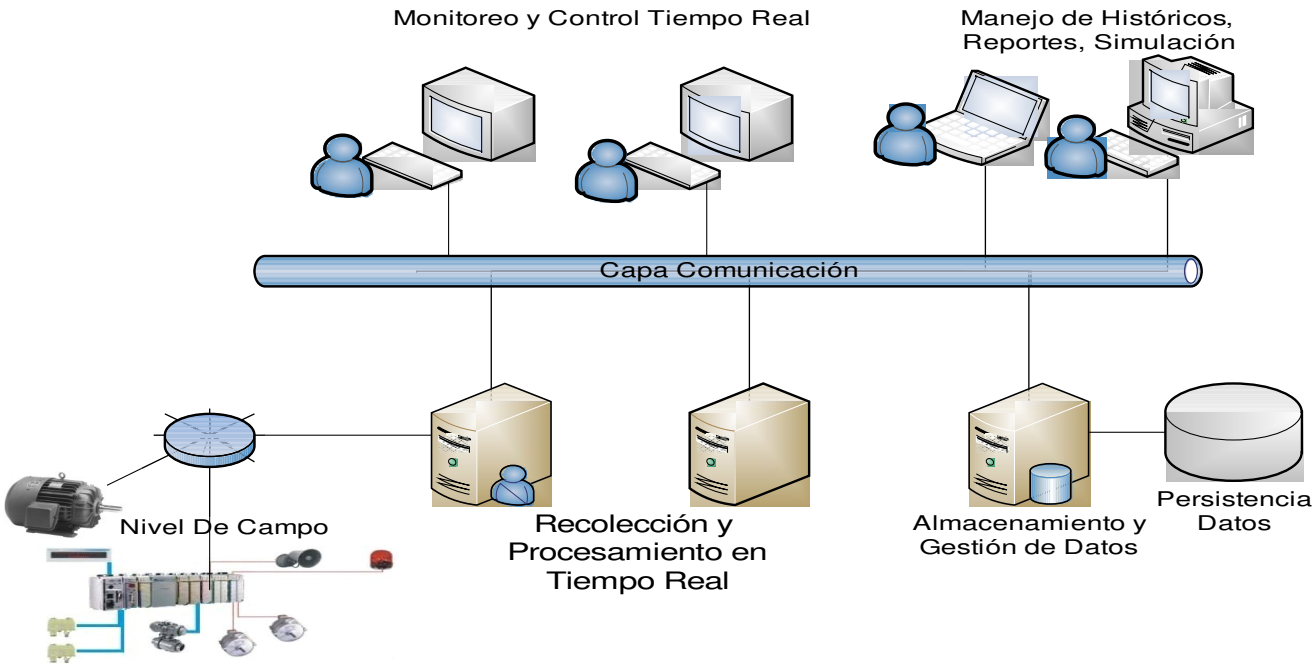


Fig. 2 Esquema general de un SCADA

1.7 Diseñadores de reportes sobre plataforma de Software Libre.

Como parte del estudio realizado de las distintas herramientas generadoras de reportes y que a su vez permitan la edición de los mismos, mostramos a continuación un resumen de las más importantes en cuanto a las funcionalidades básicas y la distribución de los componentes. De dichas herramientas se hizo un análisis en su código fuente así como en su estructura visual, con el objetivo de obtener algunas ideas y conceptos importantes.

1.7.1NCReport

Es un generador de informes que a su vez permite el diseño visual de las plantillas a generar. Tiene una apariencia visual amigable permitiendo, desde el menú principal, la utilización de los distintos componentes en la paleta de herramienta. Ejemplos de los componentes de los cuales dispone están campo de expresión, número de página, texto y etiquetas, imágenes, hipervínculo y autoforma (línea, cuadro) y tabla de referencia cruzada.(NCREPORT).

Los reportes diseñados son almacenados en archivos con formato XML permitiendo la lectura del mismo por el motor generador para la generación del reporte deseado. Esta implementado sobre C++ y Qt y bajo licencia GPL lo cual permite su uso y modificación en caso de interés para cualquier persona o entidad.

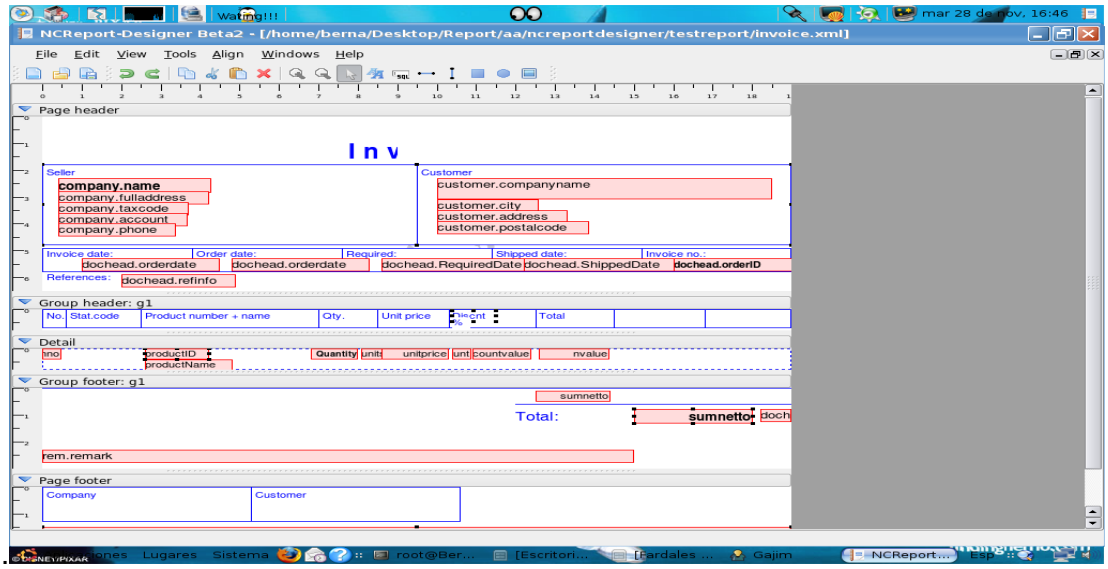


Fig. 3 Ambiente de edición del NCReport

1.7.2Report Manager

Es una aplicación de generación de reportes (Report Manager Designer). Posee una interfaz visual con buena apariencia y permite insertar componentes tales como número de página, campo de expresión, texto y etiquetas, imágenes y gráficos de barras. Este incluye una librería dinámica estándar con funciones que proporcionan una interfaz con distintos lenguajes de programación como GNU C. Este sistema es un proyecto de software libre bajo el modelo MPL (incluye permiso de uso en aplicaciones

GPL), por lo que puede ser usado en distintas aplicaciones incluyendo las que son de carácter comercial, pero cualquier mejora introducida en el motor de impresión debe ser publicada bajo esta licencia. Corre tanto en GNU/Linux como en Microsoft Windows, y dispone de generación en varios formatos de salida como son PDF, HTML y XSL. Además dispone de múltiples características, incluyendo algunas exclusivas, como varios subinformes en una página, metarchivos, fuentes de impresora, secciones externas y subinformes hijos.

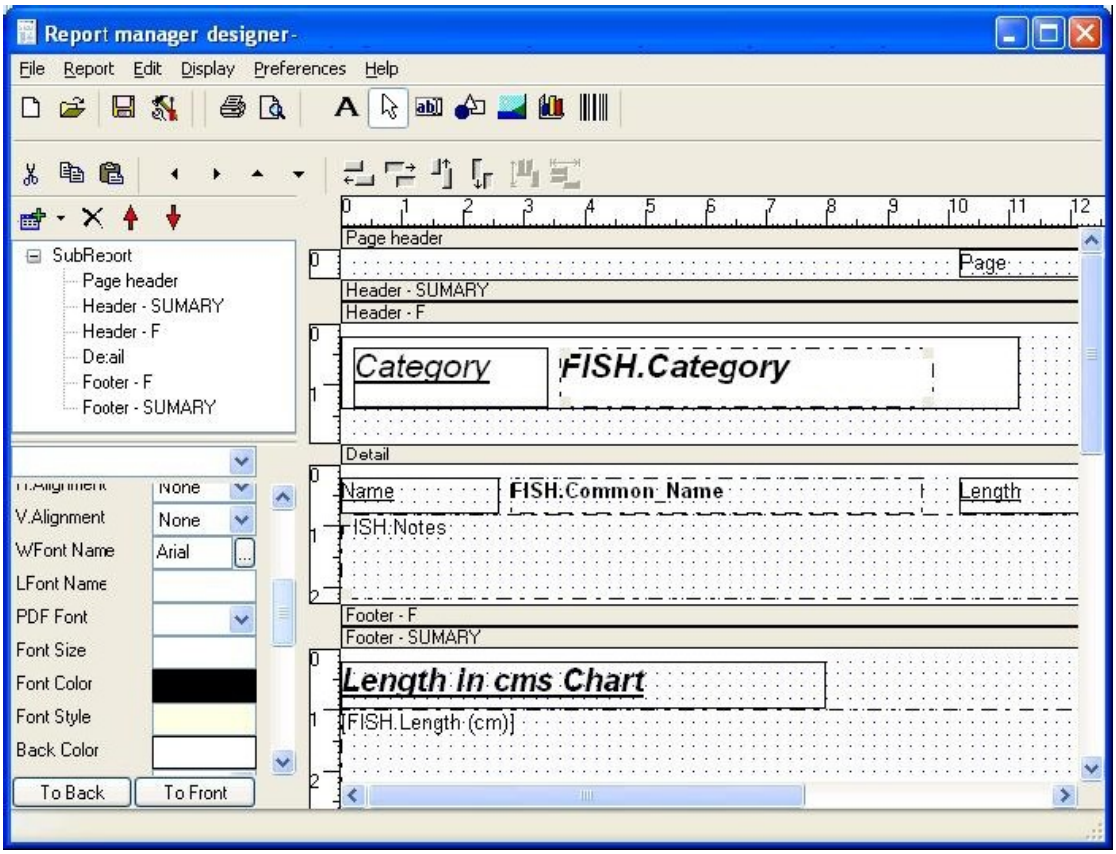


Fig. 4 Pantalla de la aplicación Report Manager Designer.

1.7.3Kugar

Kugar (KOFFICE.ORG) es una herramienta del entorno de escritorio gráfico KDE desarrollada en Qt™ para generar informes que pueden ser vistos en pantalla o impresos. No es más que uno de los componentes de Koffice, suite ofimática de KDE. Incluye un diseñador de plantillas de informes (Kudesigner), un motor, una pieza (Kpart) de Konqueror para la inspección previa fácil del informe y un

grupo de ejemplos. Esto significa que en cualquier aplicación KDE puede incluirse la funcionalidad de presentación de informes, y así, estos pueden ser vistos usando Konqueror (navegador Web y de archivos).

El diseñador de informes de Kugar es una aplicación del tipo WYSIWYG (what you see is what you get), así el tamaño de la página del informe define las dimensiones del informe en la pantalla. Mediante el uso de este diseñador se logra la creación y modificación de las plantillas de informes, y la colocación interactiva de las secciones de informe y de los elementos sobre dichas secciones. Este trae consigo toda una gama de plantillas para a partir de ellas empezar a trabajar además de poner a disposición del usuario los siguientes menús para facilitar el trabajo. La mayoría de estas opciones están disponibles en barras de herramientas para su acceso de forma más visual.

Como en muchas aplicaciones de este tipo, el área de diseño para informes puede verse dividido en varias secciones:

- *Encabezado del informe*: Define las secciones de informe que se imprimen generalmente al principio de este.
- *Encabezado de la pagina*: Define las secciones de informe que se imprimen generalmente en la parte superior de cada página del informe.
- *Encabezado de detalle*: Define las secciones del informe que se imprimen ante del detalle en un nivel dado del informe.
- *Detalle*: Define las secciones del informe que contienen los datos del mismo. El informe puede tener un número ilimitado de detalles.
- *Pie de detalle*: Define las secciones del informe que se imprimen inmediatamente después de detalles de un nivel dado.
- *Pie de la pagina*: Define las secciones del informe que se imprimen generalmente en el extremo inferior de cada página.
- *Pie del informe*: Define la sección del informe que es impresa generalmente al final del mismo.

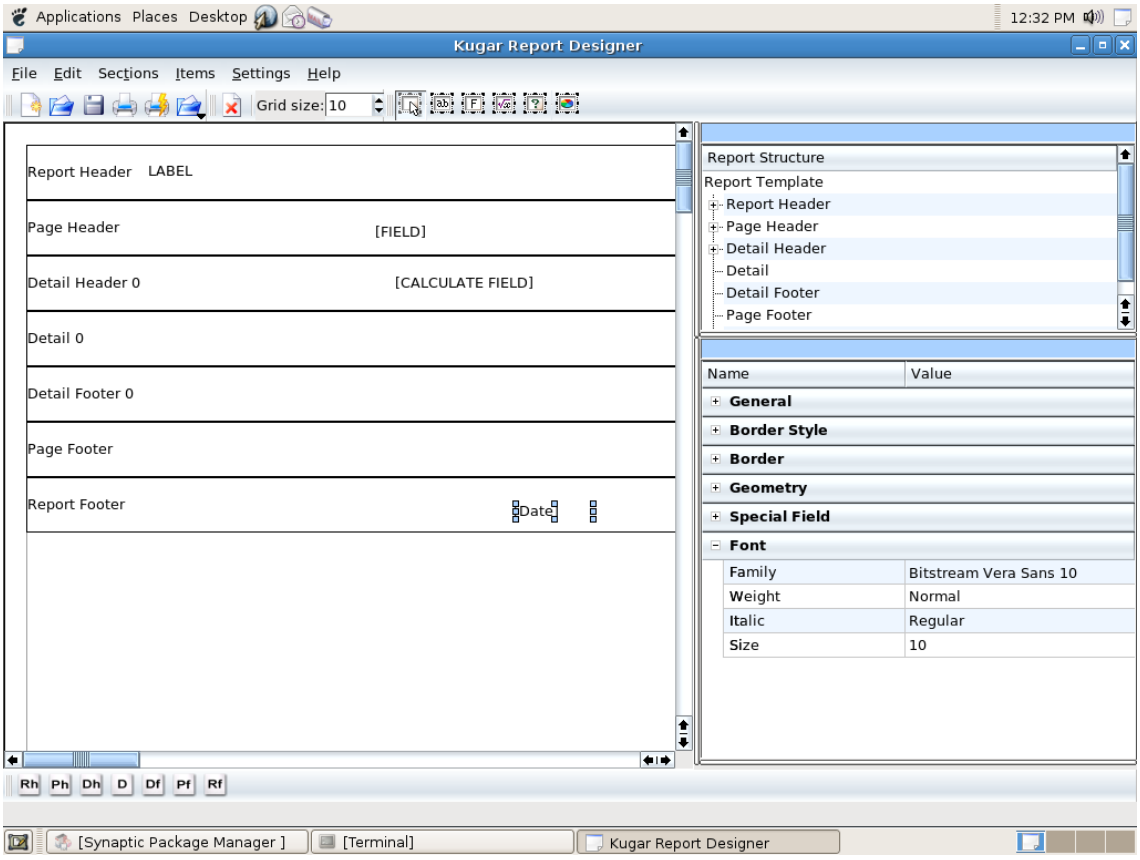


Fig. 5 Vista diseño de Kugar.

Están disponibles para la realización de diseños los siguientes elementos:

- *Etiqueta:* Un área rectangular que puede tener fronteras y se puede llenar por cualquier clase de datos textuales.
- *Campo:* Representan zonas de informaciones; sus valores serán obtenidos y procesados mientras se genera un informe.
- *Campo calculado:* Permite incluir en el informe cuentas, sumas, promedios, etc.
- *Campo especial*: Etiquetas con el texto predefinido, tal como fecha actual o número de página.
- *Líneas:* Permiten refinar el aspecto final del informe.

Tanto los elementos como las secciones tienen sus propiedades características. Esas propiedades definen parámetros geométricos, textuales y de otro tipo. Cada vez que se ubica un elemento, se aplican una serie de propiedades predeterminadas.

1.7.4Rekall

Es una herramienta muy completa la cual permite gestionar de una forma totalmente gráfica bases de datos (por ahora MySQL, PostgreSQL, xBase con XBSQL, IBM DB2 y ODBC). Así mismo, nos da la posibilidad de creación del diseño de formularios e informes, peticiones a bases de datos, importación y exportación de tablas en diversos formatos para extraer, mostrar, y editar las informaciones contenidas en estas bases de datos, de una forma muy similar al Access de Microsoft, en otras palabras, podemos decir que es una especie de Access para Linux con la diferencia de que Rekall no está ligada a ningún motor de base de datos en particular sino que se conecta a varios de los motores de bases de datos existentes. Rekall también puede diseñar y usar formularios e informes, construir consultas a base de datos, importar y exportar datos. El usuario puede crear componentes reusables y utilizarlos en formularios e informes, para reducir así, el tiempo de desarrollo de la aplicación.

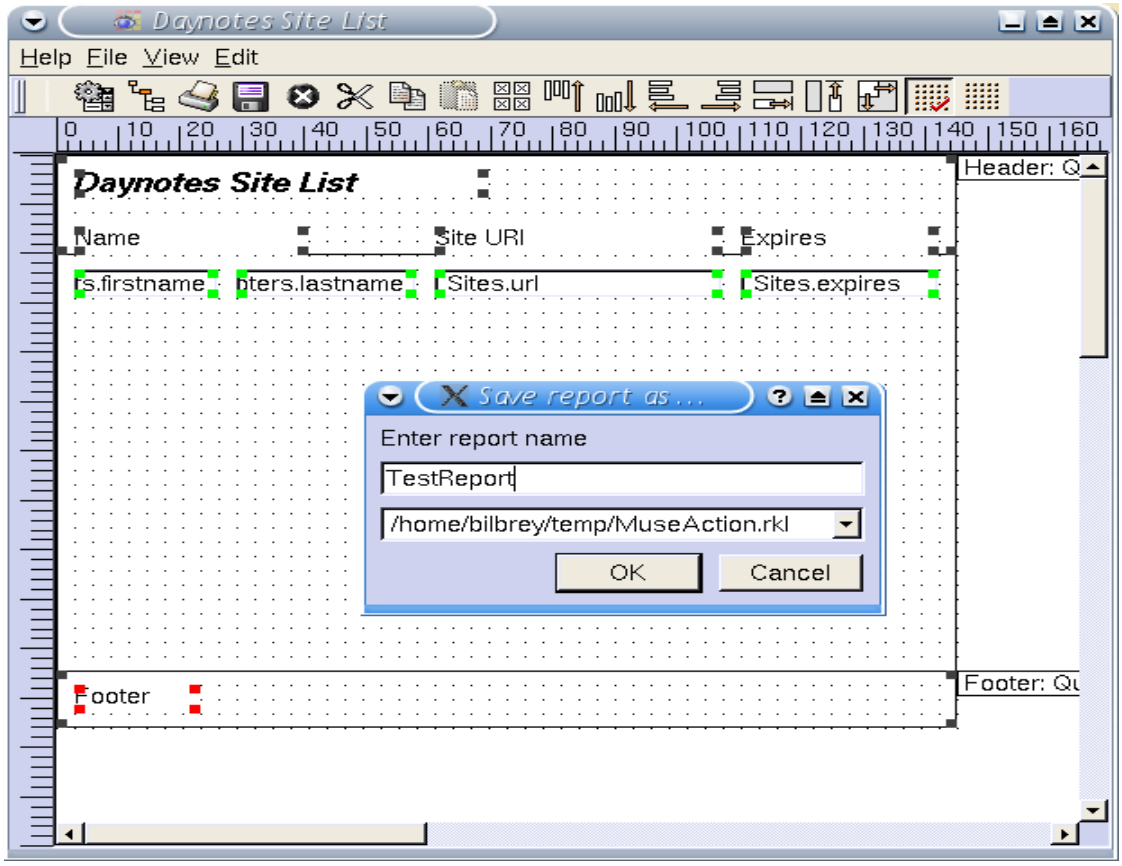


Fig. 6 Ambiente de diseño Rekall.

1.7.5Jasper Reports

Es una herramienta de fuente abierta para la generación de informes escrita en Java, que puede mostrar contenido de calidad en ficheros PDF, HTML, XLS, CSV (formatos de hoja de cálculo) y XML. Este cuenta con un diseñador con buena apariencia visual y un permite insertar numerosos componentes dada las facilidades de Java. Puede usarse en aplicaciones embebidas en Java, incluyendo J2EE (plataforma JAVA2 edición empresarial) o aplicaciones Web, para generar contenidos dinámicos. Esta librería puede usarse en Linux y en Microsoft Windows, y se distribuye bajo licencia LGPL.

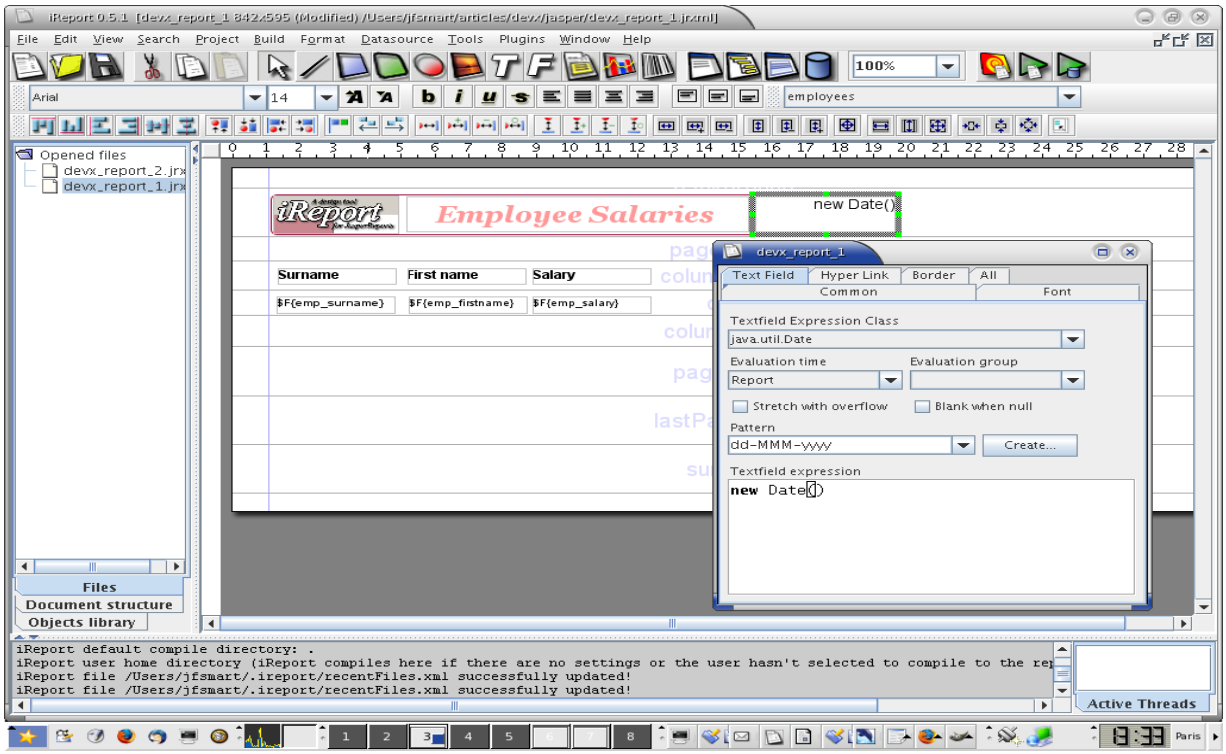


Fig. 7 Ventana de diseño de informe de JasperReports

1.7.6Data Vision

Es una herramienta de diseño y generación de informes. Los reportes pueden ser diseñados usando una interfaz gráfica que es bastante amigable. Este sistema utiliza varias de las características avanzadas que contiene JasperReports como soporte de JDBC o la disponibilidad de herramientas gráficas para la generación de informes y que a su vez añade alguna característica especial como la posibilidad de exportar los informes a formato DocBook o LaTeX2e. DataVision está hecho en Java y está soportado en múltiples plataformas. Puede generar informes tomando los datos de bases de datos o archivos de texto cuyas columnas pueden ser separadas por cualquier carácter. Puede trabajar con cualquier base de datos que tenga un controlador JDBC como son Oracle, PostgreSQL, MySQL, Informix, hsqldb, Microsoft Acces y otras. (Ver Anexo 5)

Como se puede apreciar en el anexo 5, en la filosofía de DataVision, un informe tiene múltiples secciones. En la parte izquierda están los nombres de las mismas. Las largas áreas blancas son

secciones que contienen campos del informe: columnas de la base de datos, campos de texto, fórmulas, agregaciones, y campos especiales como el título del informe o el número de la página.

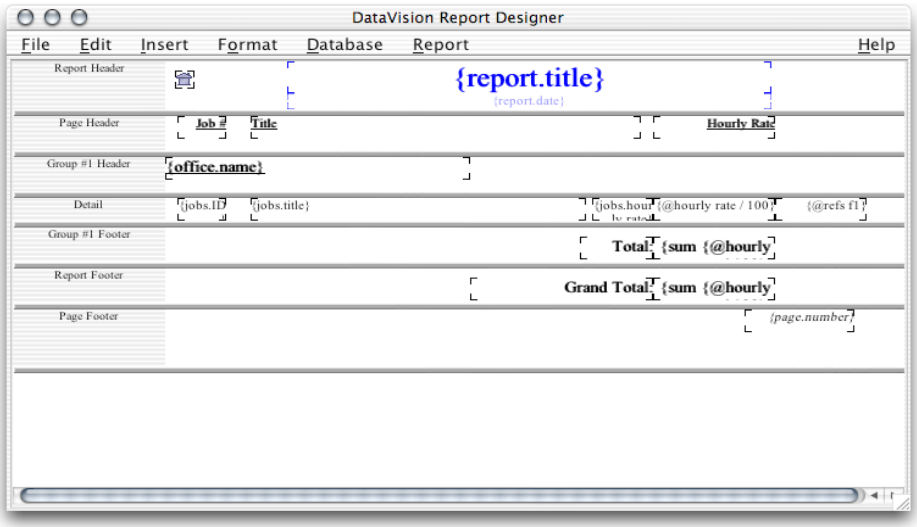


Fig. 8 Ventana de diseño de informe

1.8 Principales Tendencias y Tecnologías

Es una realidad de nuestros días las tendencias de muchas empresas, instituciones y personas civiles, que dedican tiempo a pensar en el uso o no de algunas tecnologías, con el objetivo de enfocarse y encaminarse por una de ellas, la que creen que cumple de forma soberana y equitativa los conceptos más importantes para los usuarios, como son libre distribución, código abierto, entre otros. Es por ello que cada día crece el uso de Software Libre a nivel mundial, pues en la medida de que nos damos cuenta que los software realizados sobre plataformas propietarias, se ven abarrotados de licencias, no muestran su código fuente, y el proceso de mantenimiento o cambio de versiones es realizado puramente por la compañía propietaria y en todos estos procesos se invierten grandes sumas de dinero para poder adquirir y mantener un producto determinado.

El hecho de desarrollar un sistema sobre plataforma libre es primeramente garantía de que el producto final pueda ser modificado por quien lo desee y de esta manera poder hacerle las adaptaciones o mejoras que éste estime convenientes.

1.8.1Software libre

Es todo software cuyo "código fuente", es conocido y además puede ser difundido libremente. Otro aspecto importante es que el autor de dicho software lo puede publicar y permitirle a cualquier persona o institución la modificación o el simple uso de su obra. Existen dos formas de distribuir el software, una es bajo las llamadas licencias sin restricciones de tipo BSD y la otra es bajo el uso de la GPL de la Free Software Foundation que plantea que se puede modificar el software pero sí y sólo si, el nuevo producto mantiene las mismas condiciones de uso y licencia del original y a su vez siga estando libremente al alcance de los que lo deseen.

1.8.2Metodologías de Desarrollo de Software

Las metodologías de desarrollo de software definen el cómo trabajar eficientemente evitando las problemáticas que conllevan al fracaso de muchos proyectos de software. El objetivo fundamental de una metodología es aumentar la calidad del software a producir en todas las fases de desarrollo del mismo, haciendo énfasis en la calidad y menor tiempo de construcción del software o lo que es lo mismo "producir lo esperado en el tiempo esperado y con el coste esperado" (MOLPECERES). Las metodologías de desarrollo de software se dividen en dos grupos, las llamadas "pesadas" y las que se conocen como "ágiles", como sus nombres lo indican ambos grupos tienen marcadas diferencias y la razón del uso o no de alguna de ellas está dada en la medida que el equipo de desarrollo del software determina la grandeza del producto o la simplicidad del mismo.

1.8.2.1 Metodologías "Pesadas"

Estas metodologías en el transcurso de los años han demostrado ser eficientes y muy efectivas en la realización o desarrollo de grandes proyectos de software. En las mismas se lleva bien claro la definición del proceso en su totalidad en cuanto a roles, actividades, artefactos y casos de uso. Aunque muchas personas plantean que son muy exigentes con la documentación, otras piensan que es la manera de que el producto que va creciendo tenga la calidad y la documentación necesaria.

El Proceso Unificado de Rational (RUP): Como sus siglas en inglés lo indican es Rational Unified Process. RUP es, de las metodologías existentes, una de las más generales y completas. Su surgimiento se remonta a aproximadamente treinta años atrás, lo cual nos da una medida de lo bien estructurada y refinada que está. Fue creada por James Jacobson y se puede decir que unifica los mejores elementos de las metodologías anteriores. Además está preparado para desarrollar grandes y complejos proyectos. Está enfocado al uso del paradigma orientado a objetos y utiliza a UML como su lenguaje de representación visual. Dentro de sus principales características están, que es guiado por casos de usos, que es iterativo e incremental y que centra en la arquitectura el esqueleto del producto como tal. También es válido destacar que se divide en cuatro fases importantes (Inicio, Elaboración, Construcción y Transición).

1.8.2.2 Metodologías "Ágiles"

Las metodologías ágiles brindan facilidades y soluciones que resultan en muchos casos ser las ideales para una gran cantidad de proyectos que tienen estas características. Estas metodologías se caracterizan por su sencillez, tanto en aprendizaje como en la aplicación de la misma, reduciéndose así los costos en tiempo y recursos para que un equipo determinado la aplique. Es por ello que han revolucionado su uso por parte de numerosos grupos de desarrollo de software pero hay que tener presente una serie de inconvenientes y restricciones para su aplicación, y una de ellas es que están dirigidas a equipos medianos o pequeños, donde el equipo se desarrolle en un ambiente que permita la comunicación y colaboración entre todos sus miembros durante todo el tiempo, cualquier inconveniente que presente del cliente o cualquier resistencia que emita el equipo de desarrollo puede llevar al proceso al fracaso, es por ello que el clima de trabajo, la colaboración y la relación equipo-cliente son claves para el éxito del producto final.

1.8.2.3Programación Extrema (XP):

Esta metodología llamada Programación Extrema, o Extreme Programming, es otra de las existentes en la actualidad. "Mientras que el RUP intenta reducir la complejidad del software por medio de estructura y

la preparación de las tareas pendientes en función de los objetivos de la fase y actividad actual, XP, como toda metodología ágil, lo intenta por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción." (MOLPECERES)

Es de vital importancia en su uso y aplicación que exista en todo momento un representante o miembro de la parte del cliente junto al equipo de trabajo con el objetivo de responder a cualquier duda o aclaración que necesite cualquier miembro y de esta forma no perder tiempo, además de que el software vaya creciendo justo como lo necesita y quiere el cliente. Además esta metodología basa su implantación y desarrollo en las llamadas User Stories, historias escritas por el cliente en las que describe escenarios sobre el funcionamiento del sistema y que no sólo están limitados a la interfaz de usuario, sino que también pueden describir modelos, dominio, etc.

Uno de los objetivos por los cuales apuesta esta metodología es por lograr que las iteraciones sean cortas y que las mismas generen software que el cliente puede ver.

La programación del software se realiza en dúos formados por dos desarrolladores en un ordenador. El objetivo fundamental es que cada miembro del equipo trabaje al con cada uno de los demás integrantes y con cada módulo o componente del software, de forma tal que al final todos hayan trabajado juntos y conozcan todo el producto, o al menos los detalles de cada módulo. Se trabaja sólo en las funcionalidades requeridas para la entrega más cercana y la entrega en tiempo de prototipos funcionales.

1.8.2.4Funcionalidad FDD

Sus siglas en ingles Feature Driven Development, podemos decir que esta metodología está entre XP y RUP, aunque se considera que es una metodología ligera o ágil. Está diseñada para aplicarse en proyectos que no excedan el año de producción o desarrollo y que no sean de mucha complejidad. Similar a RUP basa su proceso en iteraciones, que aunque cortas, se hacen con la idea de tener en cada una de ellas un software funcional.

Los proyectos que siguen esta metodología, deben dividirse en las siguientes fases: desarrollo de un modelo general, construcción de la lista de funcionalidades, plan de entregas sobre la base de las funcionalidades a implementar, diseño basado en las funcionalidades e implementación basada en las

funcionalidades. El trabajo de forma general se realiza en grupo y se define un responsable, que debe ser el de más experiencia, que es quien se encarga de guiar al equipo y tomar decisiones.

1.8.3UML como lenguaje de modelado

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa (Lengua de Modelación Unificada), no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la orientación a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos

1.8.4C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje multiparadigma.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT.

Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales. C++ permite trabajar tanto a alto como a bajo nivel.

1.8.5Arquitectura de tres capas.

La programación por capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

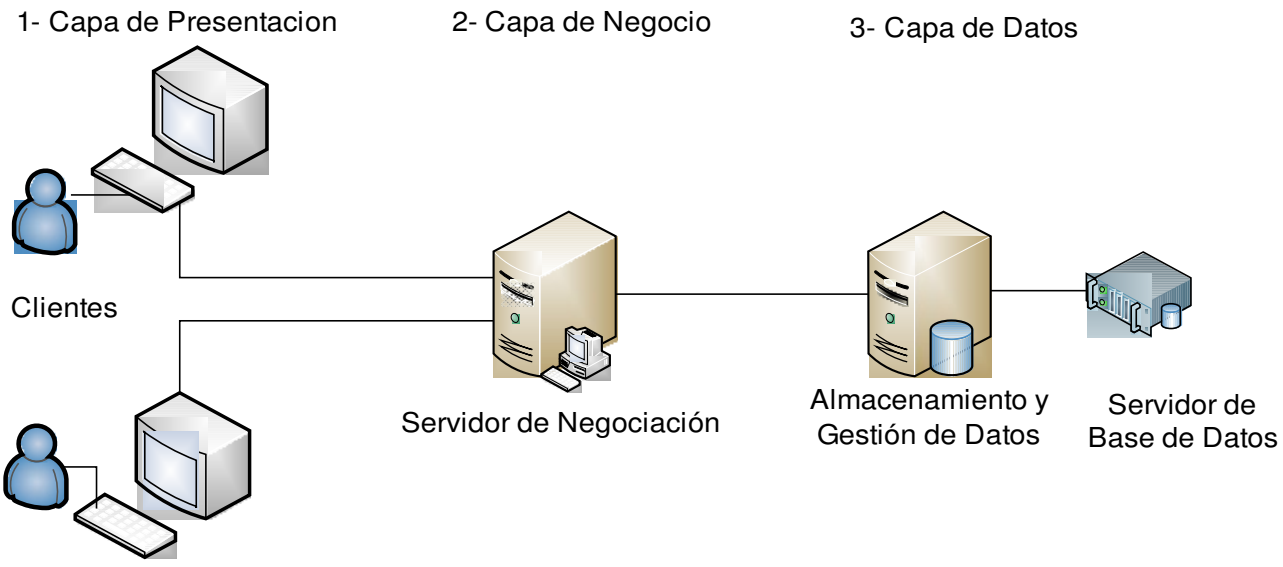


Fig. 9 Esquema Arquitectura en capas

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código

mezclado. Un buen ejemplo de este método de programación sería el modelo de interconexión de sistemas abiertos.

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

En el diseño de sistemas informáticos actual se suele usar las arquitecturas multinivel o programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

El diseño más utilizado actualmente es el diseño en tres niveles (o en tres capas).

1.8.5.1Capa de presentación

Es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario.

1.8.5.2Capa de negocio

Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

1.8.5.3 Capa de datos

Es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Todas estas capas pueden residir en un único ordenador, si bien lo más usual es que haya una multitud de ordenadores en donde reside la capa de presentación (son los clientes de la arquitectura cliente/servidor). Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o más ordenadores. Así, si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores los cuales recibirán las peticiones del ordenador en que resida la capa de negocio.

1.8.6Patrón arquitectónico Modelo Vista Controlador (MVC)

Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista.

1.8.6.1Modelo

Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos.

1.8.6.2Vista

Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

1.8.6.3Controlador

Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

1.8.7Biblioteca Qt

Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario. La biblioteca la desarrolla la que fue su creadora, la compañía noruega Trolltech, actualmente renombrada a Qt Software, y que desde junio de 2008 es propiedad de Nokia. Qt es utilizada en KDE, un entorno de escritorio para sistemas como GNU/Linux o FreeBSD, entre otros. Utiliza el lenguaje de programación C++ de forma nativa y además existen bindings para C, Python (PyQt), Java (Qt Jambi), Perl (PelrQt), entre otros.

El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales.

1.8.8Biblioteca Gráfica GTK

GTK (GIMP Toolkit) es una biblioteca multiplataforma del equipo GTK+, la cual contiene los objetos y funciones para crear la interfaz gráfica de usuario. Maneja widgets como ventanas, botones, menús, etiquetas, deslizadores, pestañas, etc.

Su licencia es la LGPL, así que mediante GTK podrá desarrollar programas con licencias abiertas, gratuitas, libres y hasta licencias comerciales no libres sin mayores problemas.

GTK está construido encima de GDK (GIMP Drawing Kit) que básicamente es un recubrimiento de las funciones de bajo nivel que debe haber para acceder al sistema de ventanas sobre el que se programe. Se llama el GIMP toolkit porque fue escrito para el desarrollo del General Image Manipulation Program (GIMP), pero ahora GTK se utiliza en un gran número de proyectos de programación, incluyendo el proyecto GNU Network Object Model Environment (GNOME).

1.8.9Cairo

Cairo es una biblioteca gráfica de la API GTK usada para proporcionar imágenes basadas en gráficos vectoriales. Aunque Cairo es una API independiente de dispositivos, está diseñado para usar aceleración por hardware cuando esté disponible. Cairo deja disponibles numerosas primitivas de imágenes de segunda dimensión.

A pesar de que está escrito en C, existen implementaciones en otros lenguajes de programación, incluyendo C++, C#, Common Lisp, Haskell, Java, Python, Perl, Ruby, Scheme (Guile, Chicken), Smalltalk y muchos otros. Dada la doble licencia incluyendo la Licencia Pública General Reducida de GNU y la Licencia Pública de Mozilla, Cairo es software libre.

1.8.10Biblioteca Boost.

Boost es un conjunto de librerías de código abierto y revisión por pares preparadas para extender las capacidades del lenguaje de programación C++. Su licencia permite que sea utilizada en cualquier tipo de proyectos, ya sean comerciales o no. Varios fundadores de Boost pertenecen al Comité ISO de Estándares C++. La próxima versión estándar de C++ incorporará varias de estas librerías.

Su diseño e implementación permiten que sea utilizada en un amplio espectro de aplicaciones y plataformas. Abarca desde librerías de propósito general hasta abstracciones del sistema operativo. Con el objetivo de alcanzar el mayor rendimiento y flexibilidad se hace un uso intensivo de plantillas. Boost ha representado una fuente de trabajo e investigación en programación genérica y metaprogramación en C++.

1.8.11IDEs

Un entorno de desarrollo integrado o, en inglés, Integrated Development Environment ('IDE'), es un programa compuesto por un conjunto de herramientas para un programador.

Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en

forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk u Objective-C.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, al que mediante pluggins se le puede añadir soporte de lenguajes adicionales.

En la actualidad existe una gran variedad de IDEs. Entre los más usados por los desarrolladores de GNU/Linux tenemos:

1.8.11.1KDevelop

KDevelop es un entorno de desarrollo integrado para sistemas Linux y otros sistemas Unix, publicado bajo licencia GPL, orientado al uso bajo el entorno gráfico KDE, aunque también funciona con otros entornos, como Gnome.

El mismo nombre alude a su perfil: KDevelop - KDE Development Environment (Entorno de Desarrollo para KDE).

KDevelop 3.0 ha sido reconstruido completamente desde los cimientos, se dio a conocer junto con KDE 3.2 en diciembre de 2003.

A diferencia de muchas otras interfaces de desarrollo, KDevelop no cuenta con un compilador propio, por lo que depende de gcc para producir código binario.

Su última versión se encuentra actualmente bajo desarrollo y funciona con distintos lenguajes de programación como C, C++, Java, Ada, SQL, Python, Perl y Pascal, así como guiones para el intérprete de comandos Bash. Además de estas características también permite el control de versiones, mantiene el completado automático de código en C y C++, tiene asistentes para generar y actualizar las definiciones de las clases y el framework de la aplicación, posee un navegador entre clases de aplicación, tiene un editor de código fuente con destacado de sintaxis e identado automático y está integrado con Qt para realizar interfaces gráficas para aplicaciones multiplataforma.

1.8.11.2Anjuta

Anjuta (actualmente Anjuta DevStudio) es un entorno integrado de desarrollo (IDE) para programar en los lenguajes C, C++, Java y Python, en sistemas GNU/Linux. Su principal objetivo es trabajar con GTK y en el escritorio GNOME, además ofrece un gran número de características avanzadas de programación. Anjuta es software libre, liberado bajo la licencia GPL.Incluye un administrador de proyectos, asistentes, plantillas, depurador interactivo y un poderoso editor que verifica y resalta la sintaxis escrita.

1.8.11.3Eclipse

Eclipse es un entorno de desarrollo integrado (IDE) de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Este incluye varias características únicas, como la refactorización de código, la actualización/instalación automática de código (mediante Update Manager). A pesar del gran número de características estándar, Eclipse se diferencia de los IDEs tradicionales en varias cosas fundamentales. Quizás lo más interesante de Eclipse es ser completamente neutral a la plataforma - y al lenguaje.

1.8.11.4Glade

Glade (o Glade Interface Designer, que significa Diseñador de interfaz Glade) es una herramienta de desarrollo visual de interfaces gráficas mediante GTK/GNOME. Es independiente del lenguaje de programación y predeterminadamente no genera código fuente sino un archivo XML (ver sección GladeXML). La posibilidad de generar automáticamente código fuente fue discontinuada desde Glade-3.

Aunque tradicionalmente se ha utilizado de forma independiente, está totalmente integrado en el recientemente liberado Anjuta 2. Cuenta con tres versiones, la primera para GTK+ 1 y las otras para GTK+ 2. Se encuentra bajo la licencia GPL.

Aunque tradicionalmente se ha utilizado de forma independiente, Glade se encuentra incorporado en el entorno integrado de desarrollo Enjuta. En su interior contiene una librería nombrada (Libglade) que mejora el uso de los archivos XML y permite una mayor flexibilidad a la hora del trabajo y trabaja sobre la licencia GPL.

1.8.11.5CxxTest

Es una biblioteca utilizada para llevar a cabo las pruebas al código fuente de cualquier proyecto. Está diseñada para ser tan portátil como sea posible. Es muy fácil de usar y de poner en funcionamiento. Dentro de sus principales ventajas están:

No requiere de funciones miembro de plantilla.

No requiere de manejo de excepciones.

No requiere ninguna biblioteca externa.

Es una herramienta multiplataforma y está disponible en los repositorios de varias distribuciones de software libre como es el caso de Debian. Se distribuye en su totalidad como un conjunto de archivos de cabecera que le hace extremadamente portátil y utilizable.

1.8.11.6Biblioteca GSL

GNU Scientific Library (GSL) es una biblioteca escrita en C, destinada a cálculos numéricos en matemáticas y ciencia, distribuida bajo la licencia GNU GPL.

Incorpora, entre otras, rutinas para el manejo de números complejos, funciones elementales y funciones especiales, combinatoria, álgebra lineal, integración y derivación numéricas, transformada rápida de Fourier, transformada wavelet discreta, generación de números aleatorios y estadística.

1.9 Tendencias para el desarrollo.

En la actualidad existen varias tendencias a la hora de enfrentar el desarrollo de productos software de este tipo. Una de ellas es la de incorporar o integrar la solución que se desea realizar a herramientas ofimáticas. Otra es la de reutilizar alguna de las herramientas generadoras de informes existentes, haciéndole las modificaciones pertinentes para llevarlo a las necesidades nuestras, lo cual trae como consecuencia que se disponga de suficiente tiempo como para entender la solución que plantea y luego comenzar a incorporarle las nuevas funcionalidades sin saber al final quede un software estable. La otra es la de desarrollar la aplicación desde cero, partiendo solo de los componentes básicos que a disposición pongan las bibliotecas que se decida emplear para el proyecto. De esta forma se dispondría de todo el código desarrollado y a la vez de toda la documentación, lo cual le daría homogeneidad a la solución total.

Aunque esta última parece que es la vía más trabajosa, es la que realmente nos da la medida de desarrollar un producto en su totalidad, además de que será más sencillo adaptarlo a las condiciones que sean necesarias, así como a la hora de intercomunicar la aplicación con otros subsistemas del SCADA sería aun más fácil puesto que conocer el diseño total de una aplicación es la ruta al éxito en la comunicación con un tercero.

1.10Propuesta

Según todo lo antes expuesto, se procede a realizar una propuesta de solución para este trabajo la cual consiste en desarrollar el diseñador de reportes usando una alternativa autónoma, donde no se pierdan

las ideas y características de otras aplicaciones similares de las estudiadas y que incluya además las nuevas funcionalidades que necesita el SCADA. El IDE que se utilizará para el desarrollo será Eclipse por ser, además de código abierto, la más potente de las herramientas de su tipo, entre las estudiadas. Además se usará el plugin para C++ ya que este va a ser el lenguaje de programación a utilizar por su portabilidad y eficiencia, así como por su alto nivel de compatibilidad con los entornos de software libre y las bibliotecas a utilizar, las cuales son: GTK como biblioteca gráfica en su versión orientada a objetos GTKmm y a su vez para la creación y tratamiento de las interfaces visuales usaremos la herramienta IDE Glade, otra de las bibliotecas es la Boost para la serialización de objetos y entidades y lograr la persistencia de los mismos, así como para usar muchas de las soluciones que brinda que son muy potentes. Además se aplicará la metodología RUP, por las ventajas que brinda y por su adecuación a proyectos de gran tamaño y duración, como es el caso, además de la flexibilidad que brinda a la hora de cambios frecuentes de requisitos. El sistema operativo donde se realizará la solución es Linux en su distribución Debian 4, por su estabilidad y por la independencia tecnológica que brinda su uso y aplicación, además de seguir el paradigma de código abierto y reutilización de código.

1.11Conclusiones.

En este capítulo se realiza un esbozo de las principales características de los generadores de reportes, además se hace un análisis de las tecnologías a utilizar en la realización de este trabajo, así como algunos conceptos y tendencias actuales que apuntan de forma exponencial al buen desarrollo de aplicaciones de este tipo. Después de haber explicado todos estos importantes aspectos se hace una propuesta con las soluciones más convenientes para enfrentar la implementación del sistema.

CAPÍTULO 2

Construcción de la solución propuesta.

En este capítulo se presentan las principales clases y funcionalidades del diseñador, relacionadas con los conceptos fundamentales que abarca, separándolos en escenarios en los cuales se explica detalladamente y se muestran los distintos diagramas del diseño. Además se evalúan los parámetros para una correcta implementación de los componentes. Para ello se describen las principales funcionalidades que deben cumplir, centrando el trabajo en las propiedades que deben manejar los mismos.

1.12Interacción del módulo Diseñador de Reportes

El funcionamiento del sistema a desarrollar básicamente consiste en permitir que la herramienta disponga de las capacidades necesarias para el diseño de informes, proporcionando al usuario, de una forma sencilla y amigable, la posibilidad de crear y diseñar una plantilla de informe o abrir una que ya exista. En esta plantilla el usuario podrá introducir imágenes, etiquetas de texto, gráficos de barra, de pastel, de línea y de puntos, además de las cajas de texto conocidas como campos de datos para indicar el origen de la información procedente de distintas bases de datos.

A continuación se muestra un esquema sencillo donde aparecen dos clientes, uno donde se está ejecutando el diseñador y el otro donde se envía al SCADA la opción de generar un reporte. El diseñador de reporte guarda la configuración o plantilla, del informe en cuestión, en el “Servidor de Configuración”, y es de esa configuración de donde se nutre el “Motor Generador” para mostrar en el “Visor” el reporte generado con la información proveniente de las distintas bases de datos volcados en su interior y con las mismas características de la plantilla que le dio origen.

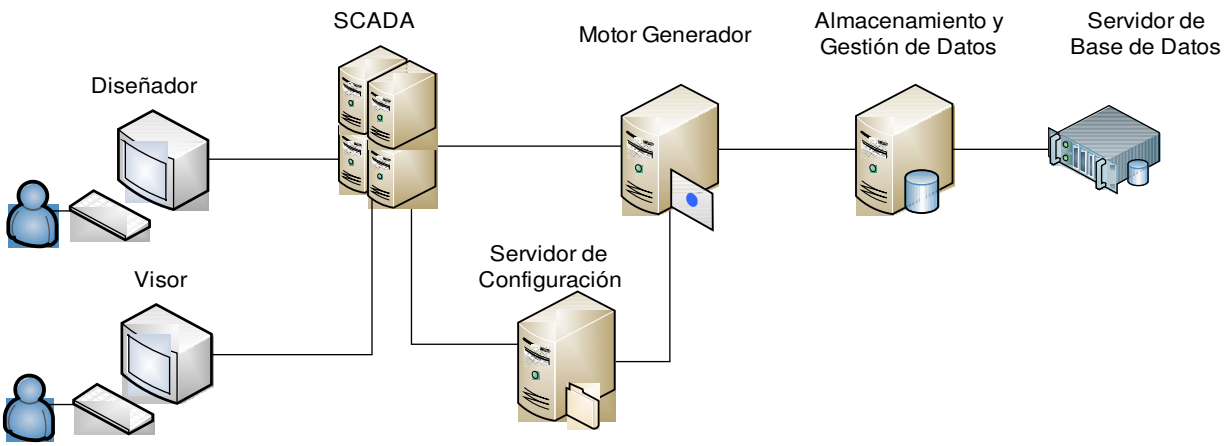


Fig. 10 Esquema de interacción del sistema propuesto con el resto de los módulos.

1.13Patrones

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

Por la similitud que existe entre el subsistema a desarrollar y una aplicación de gestión se propone una arquitectura en Vista Modelo Controlador, la cual brinda una gran flexibilidad a la aplicación dado el bajo acoplamiento funcional que se logra y proporcionando que la aplicación se adapte de forma fácil a cualquier cambio existente (adiciones, modificaciones o eliminaciones).

1.13.1Arquitectura Propuesta

La arquitectura propuesta persigue que el producto pueda dar respuesta a los requisitos de comprensibilidad, portabilidad, extensibilidad y eficiencia de forma satisfactoria. Para lograr estos fines se propone la aplicación del patrón arquitectónico Modelo-Vista-Controlador y en especial una variante del mismo, conocida como Modelo-Presentación siguiendo la arquitectura del sub-modulo del SCADA, Ambiente de Edición en el cual esta empotrado el sistema

propuesto. El uso de esta variante permite tener por separadas las responsabilidades dentro de la aplicación en dos dominios, uno sería el llamado Presentación, abarcando la lógica correspondiente a la interacción con el usuario y el otro, el llamado Modelo, en el cual se maneja la lógica perteneciente al negocio.

1.13.1.1Presentación

En el dominio o capa de Presentación es donde se implementa la interfaz de usuario y las clases que la gestionan. Estas se encargan de enlazar los eventos de la interfaz con los conocidos callbacks ó manejadores que a su vez invocarán las funcionalidades en el modelo para su encuesta y modificación.

1.13.1.2Modelo

Dentro del Modelo se identifican un conjunto de clases que se corresponden con los conceptos internos de la aplicación. Estos se asocian principalmente a estados de la configuración del informe o reporte y son independientes de cómo son representados al usuario. También se incluyen otro conjunto de clases las cuales tienen como finalidad la transformación de los estados de las distintas entidades.

1.14Funcionamiento del diseñador y relación entre los dominios.

El objetivo principal del Diseñador es brindar al usuario un ambiente amigable para la creación y diseño de las plantillas de reportes que quiera. Para lograrlo se ha implementado un conjunto de funcionalidades que debe poseer el sistema.

Según la arquitectura planteada, estas funcionalidades estarán separadas en las clases que pertenecen al modelo y las que forman parte del dominio de presentación.

Seguidamente se exponen algunos ejemplos prácticos que contribuyen a comprender la interacción entre el Modelo y la Presentación, así como la descripción de las principales clases involucradas.

1.14.1Crear Reporte

Para crear un reporte la aplicación le brinda al usuario la interfaz para realizar su petición. Está interfaz es un menú emergente que se despliega al hacer clic derecho en el árbol de proyecto. Al hacer clic en la etiqueta “Agregar Reporte” las funcionalidades de la biblioteca gráfica se encargarán de gestionar los eventos correspondientes. Estos enlazarán en la presentación con los respectivos manejadores o callbacks y los mismos se encargarán de ejecutar una función de una clase controladora del modelo (generalmente una clase de tipo Command o comando) la cual será la encargada de la modificación y encuesta al modelo. En el caso de crear reporte esta clase controladora sería *CreateReport*, que luego de haber creado el proyecto llamaría a otra controladora (en este caso la clase *ShowProperties*) encargada de mostrar las vistas (clase PropertiesInspectorReport) con los parámetros por defecto del reporte creado (Ver Figura 4), permitiendo al usuario modificar los que desee. Luego la clase controladora ShowProperties se encarga de mandar a actualizar los datos en el modelo una vez que se de la opción de salvar las propiedades del reporte en cuestión.

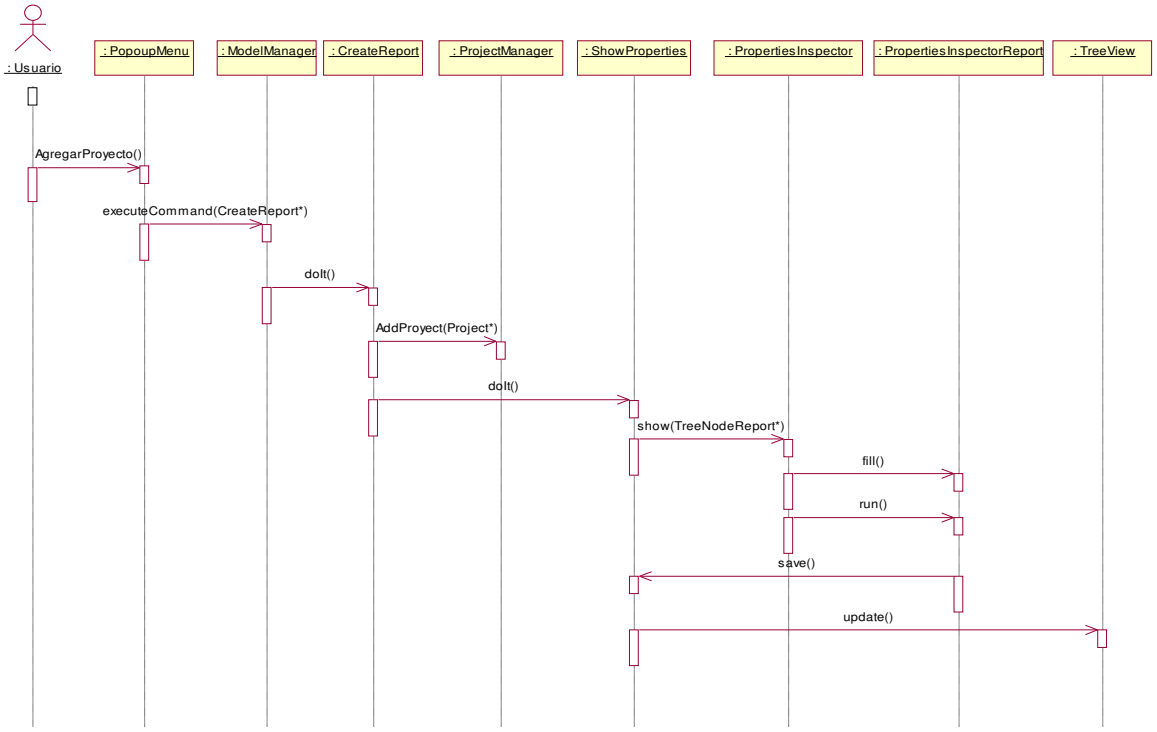


Fig. 11 Diagrama de Secuencia “Crear Reporte”

En el siguiente diagrama se muestran las clases encargadas de dar solución a la creación de un reporte

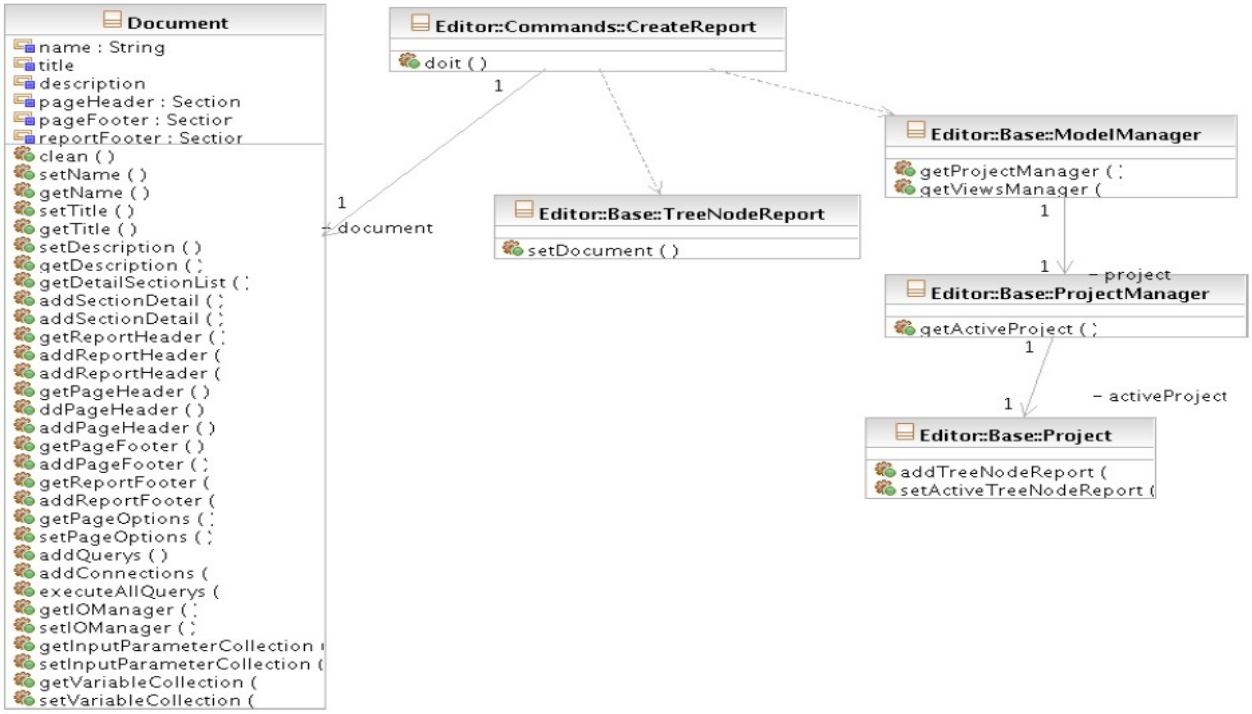


Fig. 12 Diagrama de Clases “Crear Reporte”.

1.14.1.1 Descripción de la Principales Clases

A continuación se presentan las descripciones de algunas de las clases involucradas en el escenario anterior. En el caso de las clases ProjectManager, ModelManager, Project, TreeView y PopupMenu no aparecen sus descripciones, pues ya han sido desarrolladas y detalladas en el trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas del autor Yosell Luis Sehara Driggs. Este también es el caso de la clase Document, la cual pertenece a los autores Bernardo Zaragoza Hijuelos y Raudi Agdel Bacallao Sanchez.

La clase CreateReport es de tipo Command, la cual engloba en su función doIt (ModelManager& modelManager) los elementos necesarios para crear un reporte, como son los conceptos de documento y secciones, los cuales son creados y tratados en esta función.

Tabla 1***: clase “CreateReport”***

Nombre: CreateReport	
Controladora	
Atributo	**Tipo**

ShowProperties	ShowProperties
document	Document
Para cada responsabilidad	
Nombre:	doIt(ModelManager& modelManager)
Descripción:	Ejecuta la creación de un reporte

La clase TreeNodeReport representa el reporte como tal, dentro del árbol de configuración. La misma contiene los conceptos de reporte y documento.

Tabla 2***: clase "TreeNodeReport"***

Nombre: TreeNodeReport	
Controladora	
Atributo	**Tipo**
report	Report*
templateDocument	Document
Para cada responsabilidad	
Nombre:	getDocument()
Descripción:	Devuelve el atributo document
Nombre:	getId()
Descripción:	Devuelve el Id del TreeNode
Nombre:	getName()
Descripción:	Devuelve el nombre del TreeNode
Nombre:	setDocument(Document& doc)
Descripción:	Cambia el atributo document
Nombre:	setName(string name)
Descripción:	Cambia el nombre del TreeNode
Nombre:	setId(unsigned int id)
Descripción:	Cambia el Id del TreeNode
Nombre:	getReport()
Descripción:	Devuelve el atributo report
Nombre:	setConfiguration(Node& node, ModelManager& model)

Descripción:	Cambia la configuración del nodo dado el administrador del modelo
Nombre:	getGroup()
Descripción:	Devuelve el grupo operacional de privilegios del atributo document
Nombre:	setGroup(string group)
Descripción:	Cambia el grupo operacional de privilegios del atributo document

Entre las clases controladoras de tipo comando tenemos a *ShowProperties* que es la encargada de mandar a mostrar la vista con las propiedades del recurso seleccionado.

Tabla 3***: clase "ShowProperties"***

Nombre: ShowProperties	
Controladora	
Atributo	**Tipo**
showMultiView	ShowMultiView
treeNodeReport	TreeNodeReport
inspectionableType	InspectionableType
Para cada responsabilidad	
Nombre:	doIt(SCADA::Editor::Base::ModelManager& modelManager)
Descripción:	Muestra la vista con las propiedades del recurso seleccionado
Nombre:	setInspectionableType(InspectionableType type)
Descripción:	Cambia el valor del atributo inspectionableType
Nombre:	setTreeNodeReport(TreeNodeReport* treeNodeReport)
Descripción:	Cambia el valor del atributo treeNodeReport

La clase "*PropertiesInspectorReport"* representa la vista para mostrar al usuario las propiedades de un reporte, permitiendo también ser modificadas.

Tabla 4 ***clase "PropertiesInspectorReport"***

Nombre: PropertiesInspectorReport	
Controladora	
Atributo	**Tipo**
treeNodeReport	TreeNodeReport
projectManager	ProjectManager
propertiesInspectorReportPage	PropertiesInspectorReportPage
propertiesInspectorReportSection	PropertiesInspectorReportSection
propertiesCollection	PropertiesCollection
groupSelectionDelegate	GroupSelectionDelegate

Para cada responsabilidad	
Nombre:	saveTreeNodeReport()
Descripción:	Salva la colección con todas las propiedades del reporte
Nombre:	setTreeNodeReport(TreeNodeReport* treeNodeReport)
Descripción:	Cambia el valor del atributo treeNodeReport
Nombre:	getPropertiesCollection()
Descripción:	Devuelve el valor del atributo propertiesCollection
Nombre:	setProjectManager(ProjectManager& projectManager)
Descripción:	Cambia el valor del atributo projectManager

A continuación se presentan a modo de descripción las propiedades generales de un reporte, en su creación, con la respectiva explicación, de forma práctica, de cada vista presentada. Dentro de la ventana general de todas las propiedades del reporte, aparecen las distintas sub ventanas, de las cuales se presentarán solo la “General”, “Configuración de Página” y “Secciones” ya que el resto de las mismas fueron descritas y realizadas en el trabajo de diploma del autor Ing. Yorji Pérez Hernández.

1.14.2Propiedades generales del diseño.

Cuando se crea un diseño es necesario establecer un conjunto de propiedades generales, las cuales definirán características esenciales del mismo. Si estas propiedades no se especifican por el usuario se establecerán un conjunto de valores por defecto. La clase que maneja esta ventana de propiedades generales del reporte es la anteriormente descrita “PropertiesInspectorReport”. Como se muestra en la figura 13, dentro de estas propiedades generales se encuentra la paleta General donde aparece el nombre y la descripción del diseño que se crea. Además, está ventana contiene el grupo operacional de privilegios el cual define quien tiene acceso a generar el reporte.

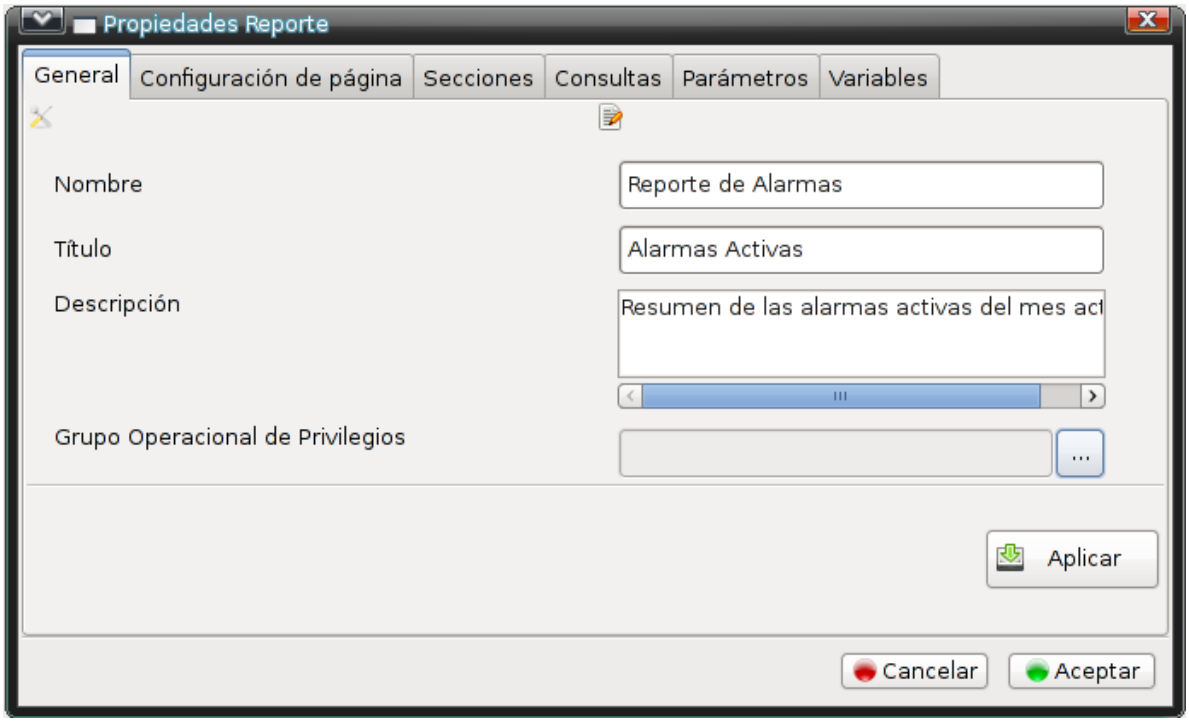


Fig. 13 Propiedades generales de un diseño.

1.14.3 Propiedades de las páginas.

Al generarse el reporte, este se desplegará sobre hojas con dimensiones estándar o personalizadas por el diseñador. En cualquiera de los casos hará falta en el momento de diseño especificar las características de estas páginas. Esto se hará a través de la interfaz mostrada en la Fig. 14. Como se puede apreciar, hará falta especificar las dimensiones de las páginas. Para facilitar este trabajo, se pueden elegir los formatos de hojas mediante la lista desplegable que aparece en el cuadro “Tamaño de hoja”.

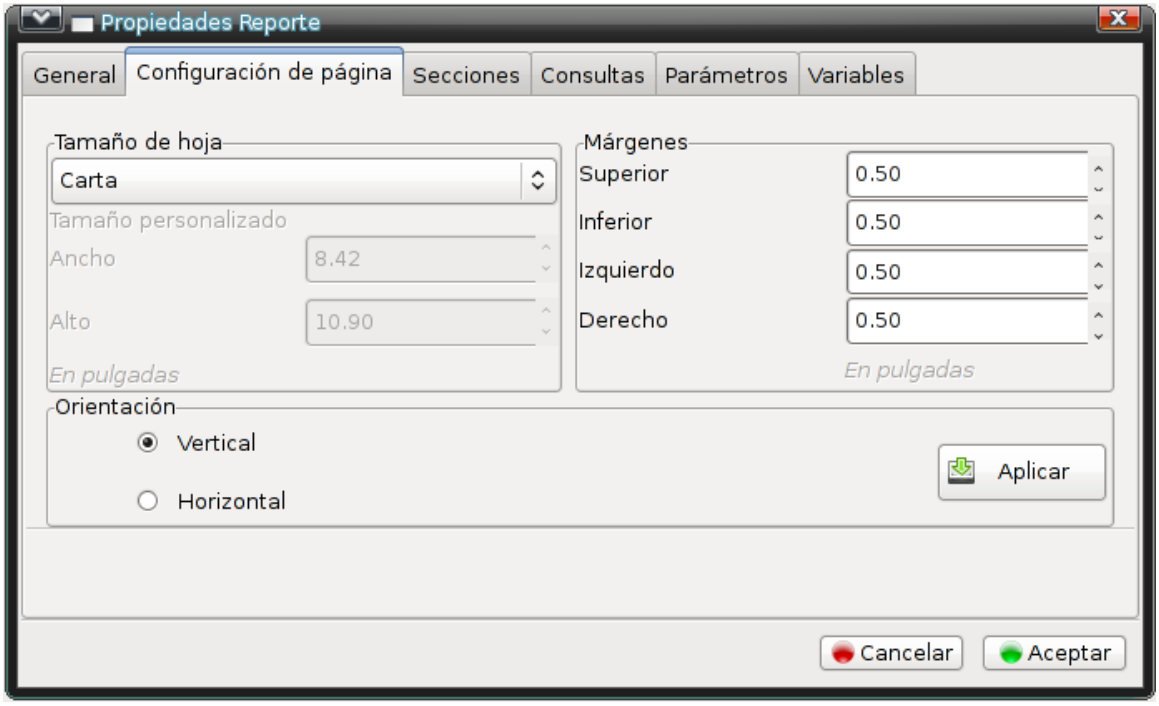


Fig. 15 Propiedades de las páginas del reporte.

Si, por el contrario, se quisiera un formato de hoja personalizado, se deberá escoger la opción "Personalizado" en este menú y bastará con especificar las dimensiones deseadas en los cuadros de edición "Alto" y "Ancho".

Además de especificar las dimensiones de las hojas se podrán especificar los márgenes laterales, así como superior e inferior en las casillas correspondientes. Esto provocará que en el momento de generar el reporte su despliegue se limite al área comprendida entre estos márgenes. Por último se podrá definir la orientación que se le dará a las hojas para desplegar sobre ellas el reporte.

La clase que maneja esta vista es "PropertiesInspectorReportPage" la cual pertenece al dominio de Presentación de la cual existe una instancia en la clase "PropertiesInspectorReport" descrita anteriormente. A continuación se presenta parte de la descripción de dicha clase pues tiene varios atributos, que son componentes visuales de GTK, de los cuales se presentan solo algunos de ellos.

Tabla 5: ***clase "PropertiesInspectorReportPage"***

Nombre: PropertiesInspectorReportPage	
Controladora	
Atributo	**Tipo**
templateReport	Document*
spinbuttonReportWidth	SpinButton*
propertiesInspectorReportPage	SpinButton*
radioButtonReportPortrait	RadioButton*
radioButtonReportLandscape	RadioButton*
buttonReportApplyPage	Button*
Para cada responsabilidad	
Nombre:	fill()
Descripción:	Carga la vista, cuando es llamada, con los datos provenientes del modelo.
Nombre:	save()
Descripción:	Devuelve al modelo una colección con todos los datos de la vista almacenados en la misma.
Nombre:	cancel()
Descripción:	Cancela la vista, la ventana se cierra sin guardar cambio alguno
Nombre:	clear()
Descripción:	Limpia todos los componentes de la ventana.
Nombre:	setDocument(Report::Base::Document& doc)
Descripción:	Cambia el valor del atributo templateReport
Nombre:	changeSizePaper()
Descripción:	Cambia el tamaño de la pagina por los nuevos valores insertados

1.14.4Secciones

Cuando se realiza un diseño una de las tareas más creativas es la especificación del comportamiento que tendrá el despliegue de los datos sobre el área disponible. Para controlar este comportamiento, en el instante de diseño se maneja el concepto de sección. Una sección no es más que un área acotada que define el espacio donde se desplegarán los datos que se obtengan en el momento de generar el reporte. La Fig. 16 muestra como el área de diseño aparece dividida en 5 fragmentos verticales separadas por líneas dobles de color gris, estas son las referidas secciones. Existen varios tipos de secciones cuya peculiaridad fundamental es su comportamiento en el momento de generación. En la figura 15 se muestra de forma esquemática las principales secciones que pueden formar parte de un diseño y además son las que aparecerán por defecto al crear uno nuevo.

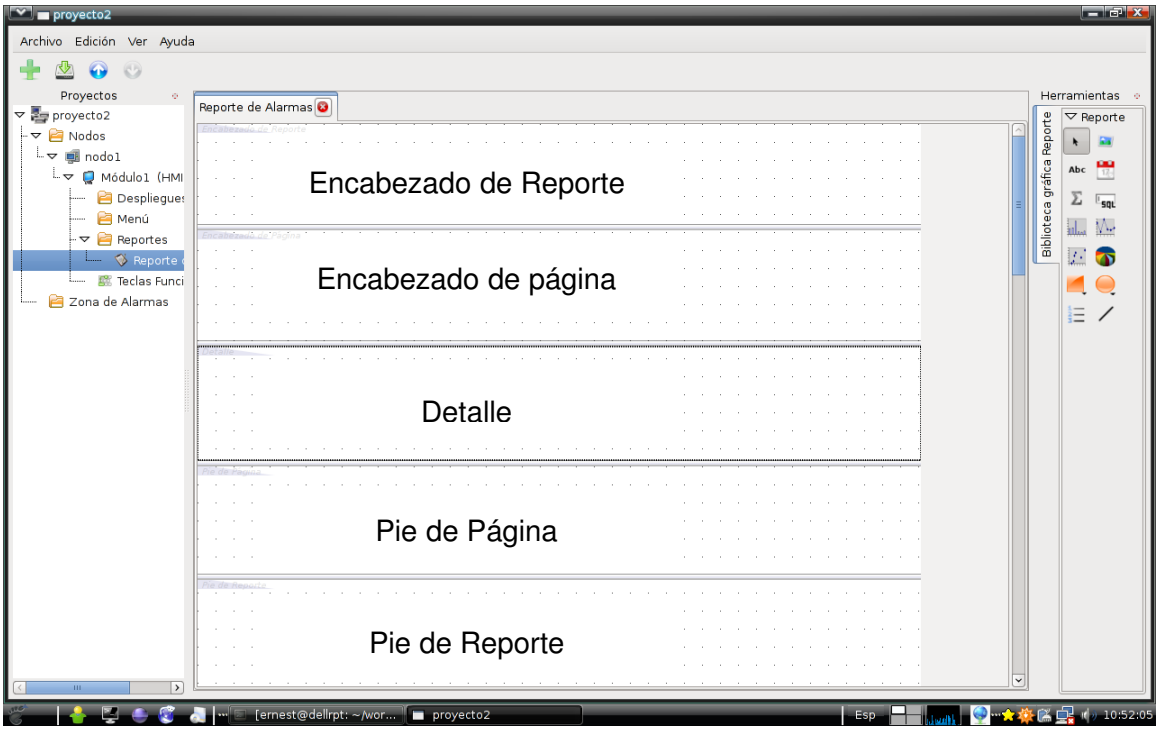


Fig. 16 Secciones básicas de un diseño.

En este esquema se puede apreciar la división del diseño en cinco secciones, el uso típico de las mismas es el siguiente:

- **Encabezado del reporte:** Es una sección que sólo se muestra en la parte superior de la primera página, normalmente se usa para especificar el título del reporte que se generará.
- **Encabezado de página:** Sección que se muestra en la parte superior de todas las hojas, normalmente empleada para colocar generalidades referidas al reporte que permitan identificar la pertenencia de una hoja a un determinado reporte como nombre del reporte, fecha de generación, etc.
- **Detalle:** En esta sección normalmente se despliegan los datos que provienen de la ejecución de consultas. En el instante de generación, esta sección se repetirá tantas veces como sea necesario para desplegar todo el flujo de datos que recibe, usando para ello el área libre que queda entre la zona de encabezados y pies, dando lugar a la generación de nuevas páginas.
- **Pie del reporte:** Es una sección que sólo se muestra en la parte inferior de la última página del reporte. Normalmente es usada para poner fin a un reporte. Suele además desplegarse en esta sección información que consolida o resume todos los datos desplegados en el reporte, típicamente totales, medias y demás funciones estadísticas.
- **Pie de página:** Sección que se muestra en la parte inferior de todas las hojas, normalmente empleada para colocar los números de página.

La descripción anterior se corresponde con el comportamiento típico de estas secciones y más adelante, dentro del escenario “Insertar Componente” se describe la clase Section como entidad del modelo. Mediante la interfaz de edición de las secciones, mostrada en la figura 16, se puede especificar otro comportamiento mediante la selección de las opciones que aparecen en la parte izquierda de esta ventana. De esta se puede inferir que en un diseño se podrán o no incluir los encabezados, pie de reporte y pie de páginas. En el caso de los encabezados y pie de página, se podrá especificar si queremos que se muestren sólo en la primera y la última página del reporte generado o si se desea que aparezcan a lo largo de todo el reporte.

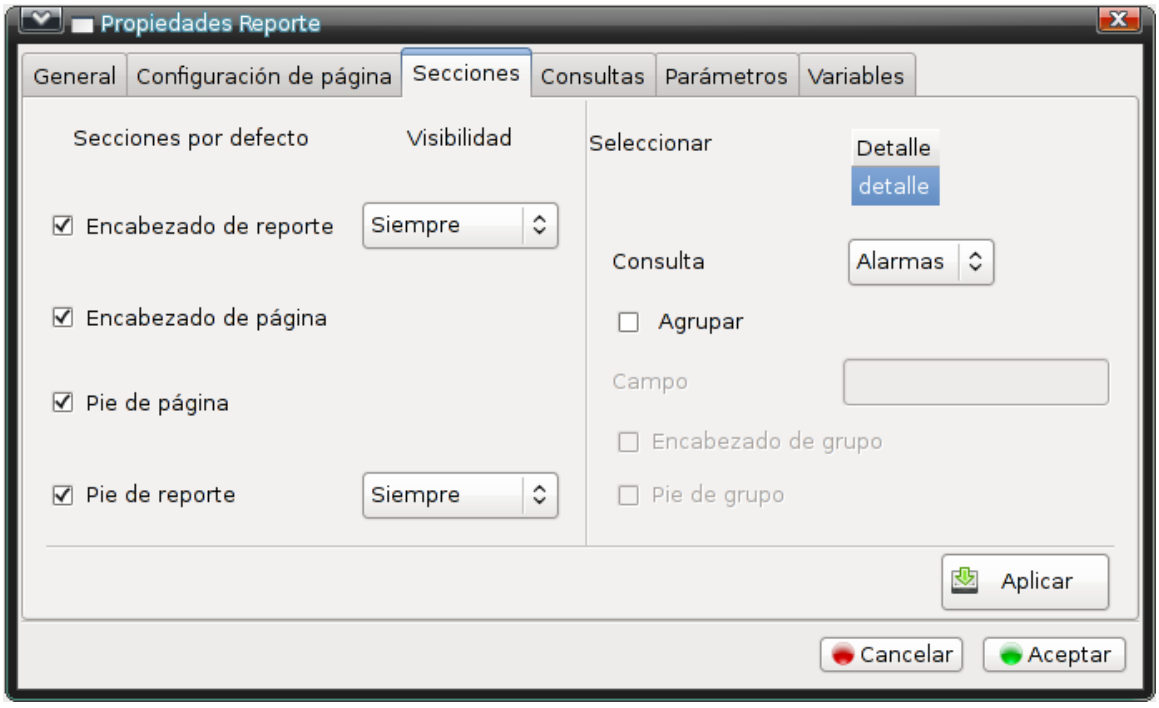


Fig. 17 Edición de Secciones.

Esta interfaz cuenta con una lista de todos los secciones de tipo “detalles” existentes en el diseño, que por su complejidad en esta versión solo concentra la aparición de un solo detalle para cada reporte. Dicha interfaz cuenta además con la lista de todas las consultas creadas, de las cuales se seleccionará la que se quiere que se muestre en el detalle del reporte generado o lo que es lo mismo, cada detalle tendrá asociada una consulta para volcar los datos en los campos de datos insertados en el reporte.

A continuación se muestra parte de la descripción de la clase correspondiente a la vista de secciones en la ventana de propiedades del reporte. Por su gran cantidad de atributos de GTK solo se describen algunos de ellos.

Tabla 6***: clase “PropertiesInspectorReportSection”***

Nombre: PropertiesInspectorReportSection	
Controladora	
Atributo	**Tipo**

templateReport	Document*
checkbuttonReportHeaderR	CheckButton*
checkbuttonReportFooterR	CheckButton*
vboxReportGroup	VBox*
entryReporFieldGroup	Entry *
buttonReportApplySection	Button *

Para cada responsabilidad	
Nombre:	fill()
Descripción:	Carga la vista, cuando es llamada, con los datos provenientes del modelo.
Nombre:	save()
Descripción:	Devuelve al modelo una colección con todos los datos de la vista almacenados en la misma.
Nombre:	cancel()
Descripción:	Cancela la vista, la ventana se cierra sin guardar cambio alguno
Nombre:	clear()
Descripción:	Limpia todos los componentes de la ventana.
Nombre:	setDocument(Report::Base::Document& doc)
Descripción:	Cambia el valor del atributo templateReport
Nombre:	addQuery(string name,unsigned int id)
Descripción:	Adiciona al ComboBox que muestra las consultas cada una de las que han sido creadas

Como se ha podido observar la primera parte de la creación de un diseño de reporte consiste en definirle una serie de características o propiedades que son de vital importancia para la visualización del reporte deseado. Una vez que ya se definieron estas características se lleva a cabo el proceso de diseño del reporte y donde el escenario fundamental es el llamado “Insertar Componente” el cual comienza cuando el usuario hace clic sobre la paleta de componentes seleccionando el que desea insertar y se activa el comando llamado “SelectTool” quien contiene el tipo de componente y se invoca una instancia de la clase CommandExecutor para que envíe al administrador del modelo llamado “ModelManager” la petición de

fabricar el objeto deseado. Una vez que ya es seleccionado el componente entonces debe hacer clic sobre el área de edición, en cualquiera de sus secciones y se muestra el componente deseado actuando el comando perteneciente al modelo “AddObject” sobre las estructuras correspondientes para que se actualice la capa de presentación “ReportEditorView”. Este es otro de los ejemplos donde se pone de manifiesto, de forma práctica, la relación entre ambos dominios, Modelo y Presentación.

Por la importancia que tienen los objetos gráficos en este trabajo, a continuación se presenta el diagrama de clases correspondiente a la jerarquía de componentes, destacando que el “Campo de Datos”, el “Campo Calculado” y los “Gráficos” ya fueron presentados y analizados, por su gran complejidad, en la tesis de grado del autor Ing. Yorji Pérez Hernandez.

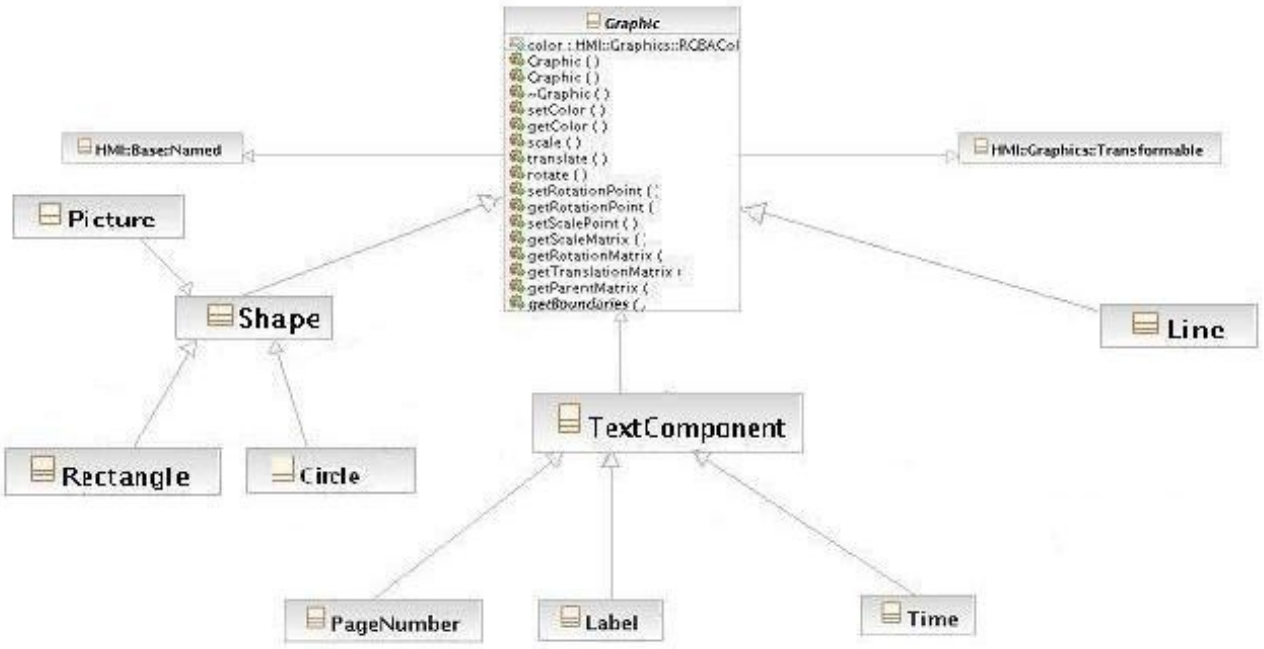


Fig. 18 Diagrama de Clases “Jerarquía de Componentes”

En el diagrama de clases anterior, figura 17, se observa como la clase base Graphic hereda además de de las clases Named y Transformable de las cuales obtiene características importantes como son nombre, identificador y la matriz de transformación. Se puede apreciar también como existen dos clases herederas de Graphic, una es Shape, clase base de Circle y Rectangle, y la otra derivada es TextComponent que es base de PageNumber, Label y Time. A continuación se muestra la descripción de las clases de esta jerarquía.

Dentro de las clases bases de la jerarquía anterior se encuentra la clase Transformable de la cual sus derivadas heredarán las funcionalidades rotar, escalar, trasladar, etc, dada su matris de tranformación.

Tabla 7***: clase "***Trasformable***"***

Nombre: Trasformable	
Entidad	
Atributo	**Tipo**
matrix	Matrix
Para cada responsabilidad	
Nombre:	scale(double x, double y)
Descripción:	Permite escalar el componente hasta el punto (x;y) pasados como parámetros
Nombre:	traslate(double x, double y)
Descripción:	Permite trasladar el componente hasta el punto (x;y) pasados como parámetros
Nombre:	rotate(Angle angle)
Descripción:	Permite rotar el objeto según el ángulo pasado como parámetros
Nombre:	getMatrix()
Descripción:	Devuelve el valor del atributo matrix

La clase Named es la entidad manejadora del nombre de cada objeto y a su vez contiene al identificador del mismo, puesto que ella deriva de la clase Object que contiene este atributo.

Tabla 8***: clase "***Named***"***

Nombre: Named	
Entidad	
Atributo	**Tipo**
name	string
Para cada responsabilidad	
Nombre:	getName()
Descripción:	Devuelve el valor del atributo name
Nombre:	setName(string name)
Descripción:	Cambia el valor del atributo name
Nombre:	getIdentifier()
Descripción:	Devuelve el valor del atributo identifier de su clase base
Nombre:	setIdentifier(int id)
Descripción:	Cambia el valor del atributo identifier de su clase base

La clase Graphic es la entidad que representa a cada objeto grafico, como se muestra en el diagrama anterior toma el comportamiento de las dos clases descritas anteriormente Named y Transformable.

Tabla 9: ***clase "***Graphic***"***

Nombre: Graphic	
Entidad	
Atributo	**Tipo**

color	RGBAColor
Para cada responsabilidad	
Nombre:	setColor(const RGBAColor&)
Descripción:	Cambia el valor del atributo color
Nombre:	getColor()
Descripción:	Devuelve el valor del atributo color
Nombre:	getBoundaries()
Descripción:	Devuelve la referencia del objeto BoundingRect de cada graphic
Nombre:	clone()
Descripción:	Devuelve un objeto Graphic idéntico al objeto en cuestión

La clase Shape abarca el concepto de aquellos objetos que son gráficos y que a la vez se puedan rellenar en su interior, como es el caso del círculo, el rectángulo y la imagen.

Tabla 10: ***clase "***Shape***"***

Nombre: Shape	
Entidad	
Atributo	**Tipo**
lineColor	RGBAColor
lineWidth	double
Para cada responsabilidad	
Nombre:	setLineColor (const RGBAColor&)
Descripción:	Cambia el valor del atributo lineColor
Nombre:	getLineColor ()
Descripción:	Devuelve el valor del atributo lineColor
Nombre:	getBoundaries()
Descripción:	Método virtual que deben implementar las derivadas para devolver la referencia del atributo boundingRect

Nombre:	getLineWidth()
Descripción:	Devuelve el valor del atributo lineWidth
Nombre:	setLineWidth(double lineWidth)
Descripción:	Cambia el valor del atributo lineWidth

La clase Circle es la entidad que modela al componente círculo.

Tabla 11: ***clase "***Circle***"***

Nombre: Circle	
Entidad	
Atributo	**Tipo**
boundingRect	BoundingRect
style	Style
Para cada responsabilidad	
Nombre:	setStyle (Style style)
Descripción:	Cambia el valor del atributo style
Nombre:	getStyle ()
Descripción:	Devuelve el valor del atributo style
Nombre:	getBoundaries()
Descripción:	Devuelve la referencia del atributo boundingRect

La clase Rectangle es la entidad que modela al componente rectángulo.

Tabla 12***: clase "***Rectangle***"***

Nombre: Rectangle
Entidad

Atributo	**Tipo**
boundingRect	BoundingRect
style	Style
finalVertex	Point
Para cada responsabilidad	
Nombre:	setStyle (Style style)
Descripción:	Cambia el valor del atributo style
Nombre:	getStyle ()
Descripción:	Devuelve el valor del atributo style
Nombre:	getBoundaries()
Descripción:	Devuelve la referencia del atributo boundingRect
Nombre:	getFinalVertex ()
Descripción:	Devuelve el valor del atributo finalVertex
Nombre:	setFinalVertex (Point point)
Descripción:	Cambia el valor del atributo finalVertex

La clase Picture es la entidad que modela al componente imagen.

Tabla 13***: clase*** *"*Picture*"*

Nombre: Picture	
Entidad	
Atributo	**Tipo**
imgType	ImageType
imageSet	ImageSet*
boundingRect	BoundingRect
style	Style
scalePercent	int
fileName	Path

isLoaded	bool
isPNG	bool
Para cada responsabilidad	
Nombre:	setImageSet (ImageSet* imageSet)
Descripción:	Cambia el valor del atributo imageSet
Nombre:	getImageSet ()
Descripción:	Devuelve el valor del atributo imageSet
Nombre:	getBoundaries()
Descripción:	Devuelve la referencia del atributo boundingRect
Nombre:	getFileName ()
Descripción:	Devuelve el valor del atributo fileName
Nombre:	setScalePercent (int scalePercent)
Descripción:	Cambia el valor del atributo scalePercent
Nombre:	loadPNG (Path path)
Descripción:	Carga una imagen de tipo PNG ubicada en el directorio pasado como parámetro.
Nombre:	loadSVG (Path path)
Descripción:	Carga una imagen de tipo SVG ubicada en el directorio pasado como parámetro.

Entre las clases mostradas en la jerarquía de componentes se encuentra TextComponent la cual es base de todos los objetos gráficos que se pintan y visualizan en forma de texto, dígase etiqueta, número de página y fecha. En el caso de la clase Label no se muestra su descripción pues contiene exactamente las mismas características y funcionalidades de su clase base TextComponent.

Tabla 14***: clase "***TextComponent***"***

Nombre: TextComponent
Entidad

Atributo	**Tipo**
text	string
size	int
fontFace	string
fontDescription	string
boundingRect	BoundingRect

Para cada responsabilidad	
Nombre:	setText (string text)
Descripción:	Cambia el valor del atributo text
Nombre:	getText ()
Descripción:	Devuelve el valor del atributo text
Nombre:	setSize(int size)
Descripción:	Cambia el valor del atributo size
Nombre:	getSize ()
Descripción:	Devuelve el valor del atributo size
Nombre:	getBoundaries ()
Descripción:	Devuelve la referencia del objeto BoundingRect
Nombre:	getFontFace()
Descripción:	Devuelve el valor del atributo fontFace
Nombre:	setFontFace(string fontFace)
Descripción:	Cambia el valor del atributo fontFace
Nombre:	getFontDescription ()
Descripción:	Devuelve el valor del atributo fontDescription
Nombre:	setFontDescription (string fontDescription)
Descripción:	Cambia el valor del atributo fontDescription

La clase Time es la entidad que modela el componente fecha, la misma basa su comportamiento en extraer del ordenador la fecha actual y mostrarla en una cadena de caracteres en el diseño.

Tabla 15***: clase "***Time***"***

Nombre: Time	
Entidad	
Atributo	**Tipo**
dayweek	int
day	int
month	int
year	int
hour	int
min	int
sec	int
Para cada responsabilidad	
Nombre:	getDate ( )
Descripción:	Devuelve la fecha completa en el formato "día # de mes de año"
Nombre:	getTime ()
Descripción:	Devuelve la hora exacta en el formato "hora : minutos AM-PM"
Nombre:	now ()
Descripción:	Devuelve la hora exacta obtenida del ordenador

La clase PageNumber es la entidad que se encarga de modelar el componente número de página, la misma se le indica en qué pagina comienza el conteo y mantiene en cada página del reporte un número accedente hasta el final del mismo.

Tabla 16: ***clase "***PageNumber***"***

Nombre: PageNumber

Entidad	
Atributo	**Tipo**
pageInit	int
pageNumber	int
Para cada responsabilidad	
Nombre:	getPageInit ( )
Descripción:	Devuelve el valor del atributo pageInit
Nombre:	setPageInit (int pageInit)
Descripción:	Cambia el valor del atributo pageInit.
Nombre:	getPageNumber( )
Descripción:	Devuelve el valor del atributo pageNumber.
Nombre:	setPageNumber (int pageNumber)
Descripción:	Cambia el valor del atributo pageNumber.
Nombre:	increment (int inc)
Descripción:	Incrementa el número en la cantidad pasada como parámetro.

Como se había explicado anteriormente el escenario Insertar Componente es uno de los principales contenidos en esta investigación y es otro de los ejemplos donde se ponen de manifiesto la relación que existe entre el modelo y la capa de presentación. Para ello se han realizado los siguientes diagramas de secuencia y de clases, así como la descripción de cada una de las clases involucradas en el mismo.

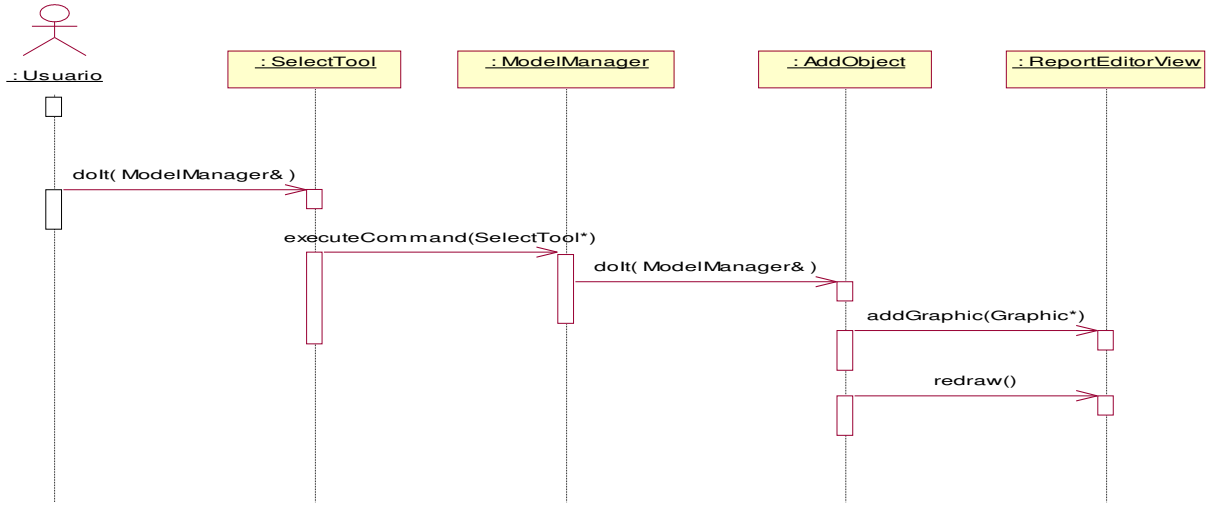


Fig. 19 Diagrama de secuencias “Insertar Documento”.

El siguiente diagrama muestra las clases involucradas en este importante escenario.

Fig. 20 Diagrama de clases “Insertar Documento”

1.14.5Descripción de las clases

Dentro del diagrama anterior están presentes algunas clases, las cuales ya han sido presentadas anteriormente y otras que ya están descritas en otros trabajos de diploma, como son ViewsManager, CommandExecutor y MultiView que están contenidas en la tesis de grado del autor Yosell Luis Sehara Driggs.

La clase AddObject es de tipo comando, y su función principal es crear el componente seleccionado y que se desea insertar, y ubicarlo en la sección deseada y en las mismas coordenadas del puntero.

Tabla 17***: clase "*AddObjet*"***

Nombre: AddObjet	
Controladora	
Atributo	**Tipo**
graphic	Graphic*
graphicClone	Graphic*
x	WindowCoord
y	WindowCoord
Para cada responsabilidad	
Nombre:	doIt(ModelManager& modelManager)
Descripción:	Ejecuta las acciones crear e insertar un componente
Nombre:	setGraphic(Graphic* graphic)
Descripción:	Cambia el valor del atributo graphic
Nombre:	setX(WindowCoord x)
Descripción:	Cambia el valor del atributo x
Nombre:	setY(WindowCoord y)
Descripción:	Cambia el valor del atributo y

La clase Section es la encargada de modelar el concepto real de sección de un reporte, que fue explicado anteriormente, y como se muestra en la siguiente descripción contiene colección de objetos gráficos que se actualiza cada vez que es insertado un componente en dicha sección.

Tabla 18***: clase "*Section*"***

Nombre: Section	
Entidad	
Atributo	**Tipo**
backgroundColor	RGBAColor
borderColor	RGBAColor
borderWidth	WindowCoord
width	WindowCoord
height	WindowCoord
yOffset	WindowCoord
graphics	GraphicCollection
name	Name
frecuency	PrintFrequency
Para cada responsabilidad	
Nombre:	setWidth(WindowCoord)
Descripción:	Cambia el valor del atributo width
Nombre:	getWidth()
Descripción:	Devuelve el valor del atributo width
Nombre:	setHeight(WindowCoord)
Descripción:	Cambia el valor del atributo heigth
Nombre:	getHeight()
Descripción:	Devuelve el valor del atributo heigth
Nombre:	getGraphicObjectCollection()
Descripción:	Devuelve el valor del atributo

Nombre:	addGraphicObject(Graphic* object)
Descripción:	Adiciona cada objeto gráfico en la sección

La clase Graphic es la entidad que representa a cada objeto gráfico o componente, esta hereda de la clase Named, de la cual recibe el atributo *nombre* e *identificador* de cada objeto.

Tabla 19***: clase "***Graphic***"***

Nombre: Graphic	
Entidad	
Atributo	**Tipo**
color	RGBAColor
Para cada responsabilidad	
Nombre:	setColor(const RGBAColor&)
Descripción:	Cambia el valor del atributo color
Nombre:	getColor()
Descripción:	Devuelve el valor del atributo color
Nombre:	getBoundaries()
Descripción:	Devuelve la referencia del objeto BoundingRect de cada graphic
Nombre:	clone()
Descripción:	Devuelve un objeto Graphic idéntico al objeto en cuestión

La clase SelectTool es de tipo comando y es la encargada de manejar la selección de cada componente en la paleta antes de ser insertado en el de diseño.

Tabla 20***: clase "***SelectTool***"***

Nombre: SelectTool	
Controladora	
Atributo	**Tipo**
selectedToolIndex	int
Para cada responsabilidad	
Nombre:	doIt(ModelManager& modelManager)
Descripción:	Activa en la paleta de componentes el componente seleccionado
Nombre:	setSelectedToolIndex(int index)
Descripción:	Cambia el valor del atributo selectedToolIndex
Nombre:	getSelectedToolIndex()
Descripción:	Devuelve el valor del atributo selectedToolIndex

La clase ReportEditorView representa la vista donde se diseña el reporte como tal.

Tabla 21***: clase "***ReportEditorView

Nombre: ReportEditorView	
Controladora	
Atributo	**Tipo**
repaintListeners	MultipleRepaintListener*
selectionRect	SelectionRect
boundaryBox	BoundaryBox
document	Document
Para cada responsabilidad	
Nombre:	update()
Descripción:	Actualiza la vista con los valores actuales de sus parámetros, ya sea después de agregaciones o eliminaciones de componentes

Nombre:	redraw()
Descripción:	Repinta toda el área dibujable con los cambios ocurridos en ese momento
Nombre:	setDocument(Document& doc)
Descripción:	Cambia el valor del atributo document
Nombre:	getDocument()
Descripción:	Devuelve el valor del atributo document
Nombre:	getBoundaryBox()
Descripción:	Devuelve el valor del atributo boundaryBox
Nombre:	getSelectionRect( )
Descripción:	Devuelve el valor del atributo selectionRect

Las clases descritas anteriormente forman parte del escenario “Insertar Componente” y precisamente son los objetos gráficos, los principales exponentes de esta importante funcionalidad del diseñador. A continuación se muestran las principales funcionalidades y propiedades de los componentes.

1.15 Funcionalidades y propiedades de los componentes.

1.15.1Componentes visuales para un diseño.

Como se refirió anteriormente para la elaboración de diseños se dispone de una paleta de componentes gráficos. Mediante una correcta disposición de estos componentes sobre las secciones del diseño se puede definir la apariencia visual de los reportes. Cada componente gráfico tiene su propio conjunto de propiedades, mediante las cuales se define su comportamiento al momento de generar un reporte. A dichas propiedades se puede acceder mediante el menú contextual que se desplegará al hacer clic derecho sobre un componente insertado en el diseño o al hacer doble clic sobre el componente. Se presenta ahora el diagrama de clases correspondiente al escenario “Mostrar Propiedades de los

Componentes" así como su respectivo diagrama de componentes y las tablas de descripción de las clases que intervienen en el mismo.

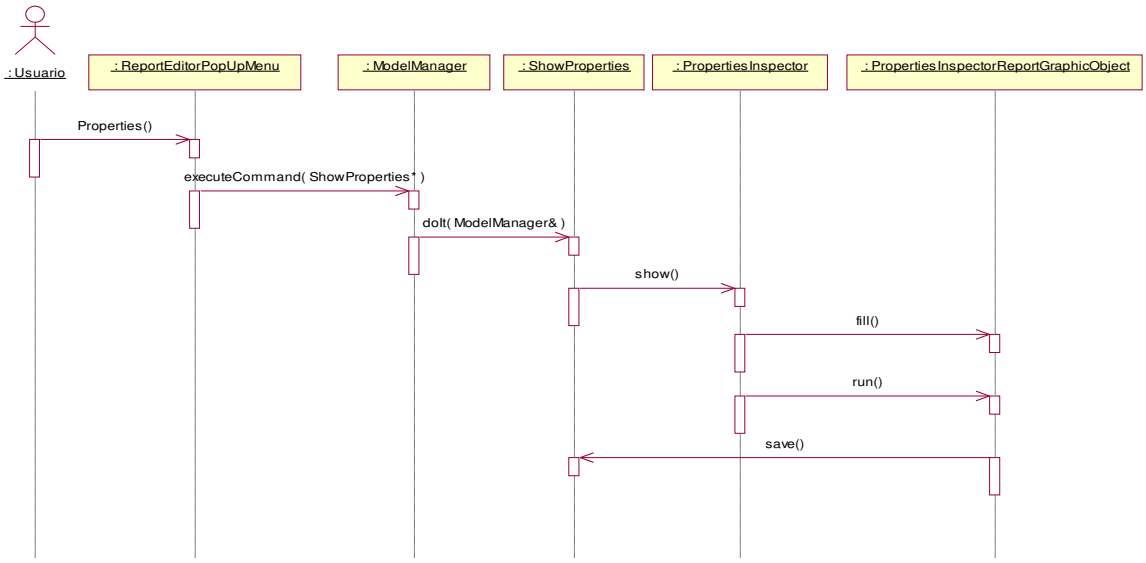


Fig. 21 Mostrar Propiedades de los componentes.

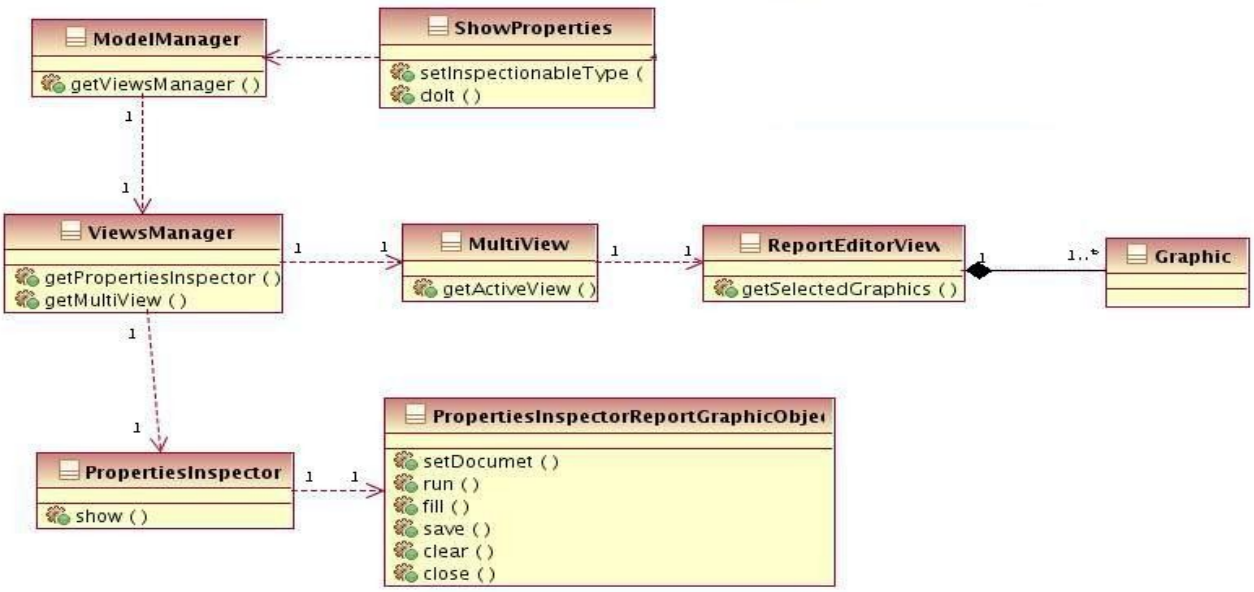


Fig. 22 Diagrama de Clases "Mostrar Propiedades de los Componentes".

De las clases relacionadas en el anterior diagrama solo PropertiesInspectorReportGraphicObject no ha sido descrita, esta clase se encarga de controlar a los inspectores de propiedades de todos los componentes gráficos existentes, en su funcionalidad run() se encarga de encuestar el objeto que está seleccionado y luego instancia al inspector de propiedades correspondiente al mismo. Seguidamente se muestra la descripción de dicha clase así como un fragmento de código de la misma.

Tabla 22***: clase "***PropertiesInspectorReportGraphicObject***"***

Nombre: PropertiesInspectorReportGraphicObject	
Controladora	
Atributo	**Tipo**
graphic	Graphic *
modelManager	modelManager*
Document	Document
propertiesInspectorReportLabel	PropertiesInspectorReportLabel*
propertiesInspectorReportLine	PropertiesInspectorReportLine*
propertiesInspectorReportPicture	PropertiesInspectorReportPicture*
propertiesInspectorReportTime	PropertiesInspectorReportTime*
propertiesInspectorReportPageNumber	PropertiesInspectorReportPageNumber*
propertiesInspectorReportRectangle	PropertiesInspectorReportRectangle*
propertiesInspectorReportCircle	PropertiesInspectorReportCircle*
Para cada responsabilidad	
Nombre:	fill()
Descripción:	Ejecuta el método fill() del inspector de propiedades correspondiente para actualizar la ventana de propiedades con los valores reales obtenidos del modelo
Nombre:	run()
Descripción:	Ejecuta el método run() del inspector de propiedades correspondiente, levantando la ventana de las propiedades del mismo
Nombre:	save()

Descripción:	Ejecuta el método save() del inspector de propiedades correspondiente, guardando la colección de datos para actualizar el modelo
Nombre:	setGraphic(Graphic& graphic)
Descripción:	Cambia el valor del atributo graphic
Nombre:	setModelManager(ModelManager& modelManager)
Descripción:	Cambia el valor del atributo modelManager

A continuación se describen las propiedades de los componentes visuales que se podrán emplear en la elaboración de un diseño y se muestran los inspectores de propiedades, de cada uno de ellos.

1.15.1.1Rectángulo y Círculo

Estos componentes son dos de las figuras básicas de las que cuenta el diseñador los cuales se utilizan para darle un mayor sentido y organización al reporte final, para mejorar la apariencia del mismo y además suelen usarse como marcos para resaltar alguna información o para conformar tablas en los reportes en el caso del rectángulo. A continuación se presentan las siguientes propiedades para definirlos:

- Nombre: Nombre que identificará al componente en el diseño, aunque este no es visible, es un nombre para deferencias las distintas instancias de este componente.
- Posición X: Coordenada en el eje de horizontal correspondiente a la posición real que tiene el componente dentro de la sección en la cual esta dibujado.
- Posición Y: Coordenada en el eje de vertical correspondiente a la posición real que tiene el componente dentro de la sección en la cual esta dibujado.
- Largo: Extensión horizontal del componente.
- Ancho: Extensión vertical del componente.
- Tipografía: Texto con los atributos del tipo de letra seleccionado. Si hacemos clic izquierdo sobre el botón de la derecha, aparecerá una interfaz gráfica que permitirá

seleccionar los posibles tipos de letra.

- Color de línea: Color que tomará la línea que bordea al componente.
- Color de fondo: Color que tomara el fondo del componente.
- Estilo de línea: Estilo que tomará la línea que bordea al componente.

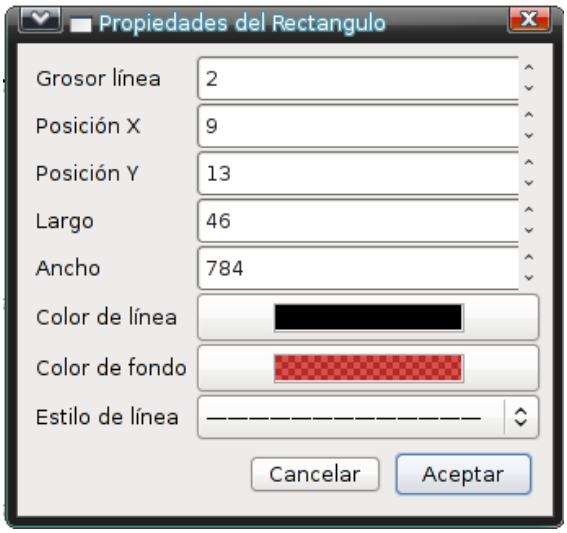


Fig. 23 Interfaz para establecer las propiedades del Rectángulo.

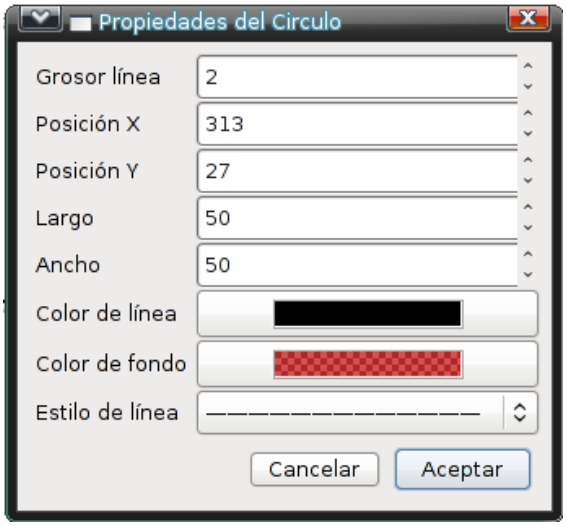


Fig. 24 Interfaz para establecer las propiedades del Círculo

1.15.1.2Etiqueta, Fecha y Número de Página

Estos tres componentes forman parte de la misma herencia y por ello presentan características similares y son muy importantes en el diseño de informes.

La etiqueta es fundamental pues es donde se plasman todos los textos del diseño. Por su parte el componente fecha facilita la obtención del día, mes, año, hora, minutos y segundos, obtenidos del ordenador, sin que el operador se preocupe por el día que es, además permitiendo que la plantilla de reporte sea útil en cualquier fecha, pues el componente en el diseño no carga los datos, solo en ejecución, a la hora de generar el reporte es donde se actualiza la fecha de ese momento. Además está el Número de Página al cual se le define en el diseño el número donde quiere el usuario que comience el enumerado de las páginas del reporte y este se va incrementando, en tiempo de generación, una unidad por cada página que tenga el reporte final. Las propiedades que deberán ser establecidas para estos componentes son las siguientes:

- Posición X: Coordenada en el eje de horizontal correspondiente a la posición real que tiene el componente dentro de la sección en la cual esta dibujado.
- Posición Y: Coordenada en el eje de vertical correspondiente a la posición real que tiene el componente dentro de la sección en la cual esta dibujado.
- Largo: Extensión horizontal del componente.
- Ancho: Extensión vertical del componente.
- Tipografía: Texto con los atributos del tipo de letra seleccionado. Si hacemos clic izquierdo sobre el botón de la derecha, aparecerá una interfaz gráfica que permitirá seleccionar los posibles tipos de letra.
- Color: Representa el color que tomará el texto del componente.

Además de estas características o propiedades, el componente Etiqueta tiene una llamada “Texto” la cual recoge el texto que va a tener plasmado el componente en el diseño y luego en el reporte final. Otro que tiene una nueva propiedad es el Número de Página y la misma es “Numero inicial” en la cual se define el número en el cual se quiere que comience el componente en tiempo de generación, y es válido aclarar que cuando se genera el reporte solo aparece el número correspondiente en el lugar donde fue insertado el componente.

Fig. 25 Interfaz para establecer las propiedades de la Etiqueta.

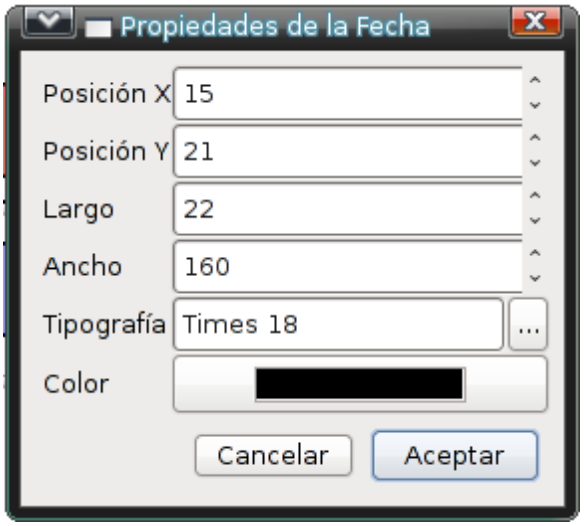


Fig. 26 Interfaz para establecer las propiedades de la Fecha

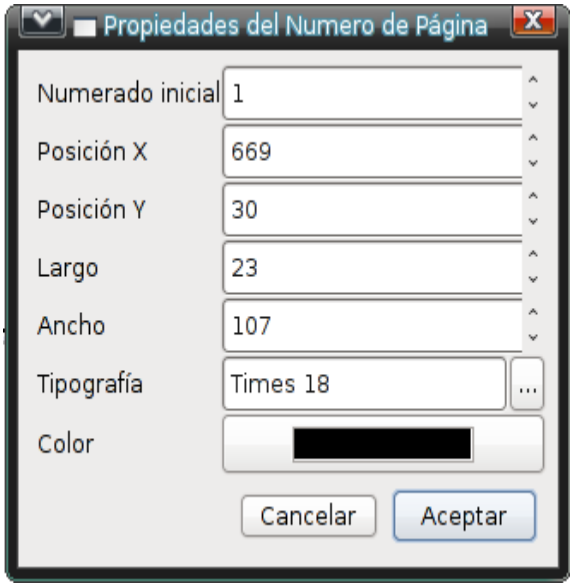


Fig. 27 Interfaz para establecer las propiedades del Número de Página

1.15.1.3Imagen

En los reportes suelen usarse imágenes que identifican la institución o que sencillamente contribuyen estéticamente a la calidad final del reporte. Para ello el diseñador proporciona un componente mediante el cual se pueden insertar en el diseño imágenes almacenadas en distintos formatos estándares cómo png y svg. Este componente, a diferencia de los restantes, se carga desde un directorio local y se dibuja con las mismas características del archivo que está siendo cargado. A continuación se presentan las siguientes propiedades para definirlo:

- Cargar Imagen: Es un componente que a su vez contiene la propiedad camino o directorio de la imagen cargada, al hacer clic sobre el mismo se despliega el buscador de archivos permitiendo buscar y seleccionar una imagen en caso de haber insertado una equivocada.
- Posición X: Coordenada en el eje de horizontal correspondiente a la posición real que tiene el componente dentro de la sección en la cual esta dibujado.
- Posición Y: Coordenada en el eje de vertical correspondiente a la posición real que tiene el componente dentro de la sección en la cual esta dibujado.
- Largo: Extensión horizontal del componente.
- Ancho: Extensión vertical del componente.

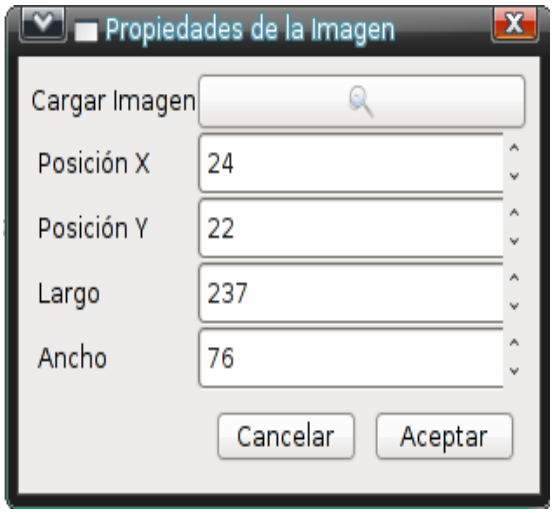


Fig. 28 Interfaz para establecer las propiedades de la Imagen

1.15.1.4Línea

El uso de este componente permite insertar líneas en un diseño. Comúnmente en los reportes las líneas son usadas como separadores. La línea es un componente que por su complejidad fue necesario separarlo de los demás, en cuanto a la distribución jerárquica de las clases, y a continuación se presentan sus propiedades.

- Grosor de Línea: Grosor que tendrá la línea al dibujarse, por defecto se asume con valor 2 para su fácil manipulación.
- Orientación: Puede ser Horizontal y/o Vertical y define como se comportará la línea en el diseño.
- Posición X inicial: Coordenada en el eje de horizontal correspondiente a la posición real que tiene el vértice izquierdo/superior del componente dentro de la sección en la cual esta dibujado.
- Posición Y inicial: Coordenada en el eje de vertical correspondiente a la posición real que tiene el vértice izquierdo/superior del componente dentro de la sección en la cual

esta dibujado.

- Posición X final: Coordenada en el eje de horizontal correspondiente a la posición real que tiene el vértice derecho/inferior del componente dentro de la sección en la cual esta dibujado.
- Posición Y final: Coordenada en el eje de vertical correspondiente a la posición real que tiene el vértice izquierdo/inferior del componente dentro de la sección en la cual esta dibujado.
- Color de Línea: Representa el color que tomará el componente.
- Estilo de Línea: Contiene el estilo que se le dará al componente.

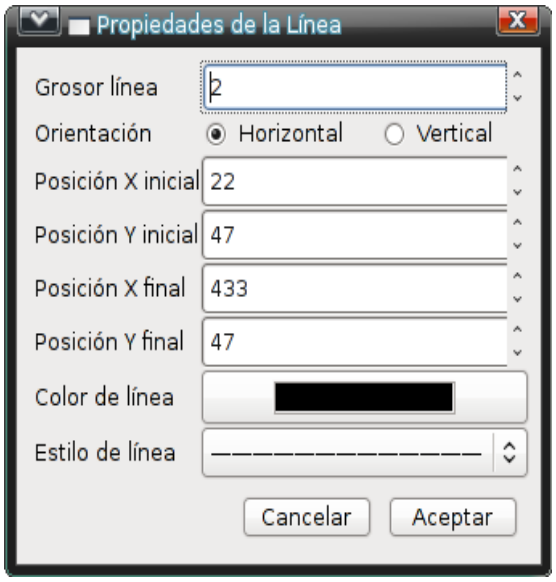


Fig. 29 Interfaz para establecer las propiedades de la línea

En todo sistema de edición o configuración es necesaria la existencia de funcionalidades básicas que faciliten el diseño. Entre las más conocidas están copiar, pegar, eliminar, duplicar, traer al frente y enviar al fondo. Todas las mencionadas han sido implementadas en esta versión. Se presentan las distintas clases relacionadas con estas acciones, así como su respectiva descripción, explicación y algunos fragmentos de código para lograr un mayor entendimiento de las mismas.

1.15.2 Funcionalidades en el diseño

1.15.2.1Copiar - Pegar

Estas dos operaciones son de las más difundidas en todo software de este tipo, permite la reutilización de uno o más componentes que ya han sido insertados sin necesidad de volver a hacerlo. Para darle solución a esta problemática fue necesario implementar una clase de tipo comando llamada CopyPaste, capaz de crear uno o varios componentes idénticos a los originales que han sido seleccionados para la "copia" y se insertan justamente en la sección que se active para "pegar" y en las mismas coordenadas del puntero del mouse. Se muestra

Tabla 23***: clase "***CopyPaste***"***

Nombre: CopyPaste	
Comando	
Atributo	**Tipo**
moveObject	MoveObject*
action	Action
xCoord	int
yCoord	int
Para cada responsabilidad	
Nombre:	doIt(ModelManager& modelManager)
Descripción:	Si la acción es copiar, Almacena en el clibBoardManager el o los componentes seleccionados para la copia, y si es pegar realiza la operación inversa, los extrae clonados y los dibuja.
Nombre:	setAction(Actio action)
Descripción:	Cambia el valor del atributo action
Nombre:	getAction()
Descripción:	Devuelve el valor del atributo action
Nombre:	setX(int xCoord)
Descripción:	Cambia el valor del atributo xCoord
Nombre:	setY(int yCoord)

Descripción:	Cambia el valor del atributo yCoord

1.15.2.2Eliminar

Para la implementación de esta funcionalidad fue necesario crear una clase, también de tipo comando, que se encargara de chequear el o los elementos seleccionados, y sacarlos de la lista de objetos gráficos y luego enviar señal de repintar la escena, ahora sin estos componentes.

Tabla 24***: clase "***DeleteObject***"***

Nombre: DeleteObject	
Comando	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre:	doIt(ModelManager& modelManager)
Descripción:	Obtiene el o los elementos seleccionados y los extrae de la lista de componentes de la vista, luego repinta la escena.

1.15.2.3 Duplicar

Esta opción es muy cómoda, pues logra duplicar uno o varios objetos, previamente seleccionados, logrando hacer una copia fiel de los mismos. En este caso los elementos duplicados se dibujan con una breve variación en las coordenadas de los originales. Similar al CopyPaste, con la diferencia que no se necesita dar la opción pegar, es instantánea la acción que ocurre, justamente cuando se selecciona la opción "Duplicar" del menú se ejecuta esta funcionalidad sin necesidad de otra acción como es pegar.

Tabla 25***: clase "***DiplicateObject***"***

Nombre: DiplicateObject	
Comando	
Atributo	**Tipo**

Para cada responsabilidad	
Nombre:	doIt(ModelManager& modelManager)
Descripción:	Almacena en el clibBoardManager el o los componentes seleccionados para la copia, luego los extrae clonados y los dibuja.

1.15.2.4Traer al frente y enviar al fondo

De la forma en que se dibujan los componentes en este trabajo, resulta muy sencillo darle solución a esta importante funcionalidad, y es precisamente debido a que a medida que se inserta un nuevo objeto en el área de pintura, este se ubica detrás de la posición que ocupa el objeto anterior en la lista de componentes de esa sección. Para ello se creó una clase llamada DepthCommand la cual es un comando y los que hace es en su método doIt() encuestar el valor del atributo type, que se refiere al tipo de acción, y si es Traer al frente busca el o los elementos que han sido seleccionados, realiza un clon de los mismos, los extrae de la lista de objetos de esa sección y luego inserta los elementos que fueron clonados en las primeras posiciones de la lista y se repinta la escena. En caso de ser “Enviar al fondo” se realiza la operación inversa.

Tabla 26***: clase “***DepthCommand***”***

Nombre: DepthCommand	
Comando	
Atributo	**Tipo**
type	Type
Para cada responsabilidad	
Nombre:	doIt(ModelManager& modelManager)
Descripción:	Obtiene el o los elementos seleccionados, realiza un clon de los mismos, los extrae de la lista de componentes de la vista, encuesta al atributo “type”, si la opción es “bringtofront” inserta los elementos colonados en las primeras posiciones de la lista y si la opción es “sendtoback” hace lo mismo, pero los adiciona al final de la lista, luego repinta la escena.
Nombre:	setType (Type type)

Descripción:	Cambia el valor del atributo “type”

1.16Conclusiones

Es de vital importancia, para todo sistema SCADA, la presencia de un diseñador de informes, esto hace más cómodo y entendible el proceso de toma de decisiones y la organización de la información. Estos subsistemas deben ser muy sencillos de usar y de fácil instalación y puesta en funcionamiento, de esta forma se logra una aplicación más robusta y a la vez permite una mayor facilidad de mantenimiento. En este capítulo se ha realizado la descripción de la mayoría de los elementos implementados, se han seleccionado las principales funcionalidades y se ha descrito detalladamente las clases más significativas que intervienen en cada uno de los escenarios propuestos.

CAPÍTULO 3

Validación de la Solución

Introducción

Todo sistema de software debe ser probado antes de su entrega o puesta en funcionamiento, a este proceso o fase se le denomina prueba o testeo de la aplicación. Es una de las etapas, en la vida del software, más importante y que en ocasiones menos recursos se le asignan, y esto es una mala práctica que conlleva en muchos casos al fracaso del producto o a la no satisfacción del cliente. No siempre las pruebas garantizan que el software esté al total esperado en aspectos funcionales, pero si detectan numerosas fallas que el desarrollador no encontró y que ahora pueden ser corregidas y vueltas a probar. En el presente capitulo se presentan las pruebas para validar el correcto funcionamiento del sistema.

3.1 Prueba de Unidad

Podemos definir el concepto de prueba de unidad o unitaria como la actividad de probar el funcionamiento de un módulo software (código) con el objetivo de lograr su correcto funcionamiento. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. Existen herramientas capaces de propiciar métodos potentes de probar funcionalidades de programas, dotadas de bibliotecas con un conjunto de clases específicamente para pruebas unitarias.

Las pruebas de unidad permiten ser ejecutadas de forma independiente, logrando que el sistema completo pueda ser dividido en varios casos de prueba y no necesariamente realizar una sola prueba de unidad a todo el software. La idea es escribir casos de prueba para cada función no trivial, o método, en el módulo, de forma que cada caso sea independiente del resto.

El objetivo fundamental de las pruebas unitarias es aislar cada parte del programa y demostrar que cada una de las partes, separadas en fragmentos individuales, son correctas. Las pruebas por lo general son bien documentadas garantizando que el software que cumpla con las pruebas diseñadas en los distintos casos de pruebas es un software con importantes parámetros de calidad.

Las pruebas unitarias, de forma general tienen las siguientes ventajas:

1. **Fomentan el cambio**: Facilitan al desarrollador la posibilidad de cambiar el código para mejorar su estructura y a la vez permiten que los nuevos cambios sean probados y así determinar que ya se llegó al fin del caso de prueba en cuestión.
2. **Simplifican la integración**: Crean un ambiente muy amigable a la hora de entrar en la fase de integración, haciendo de esta una actividad más fácil y segura y a la vez logrando que las pruebas en esta fase sean más fáciles de elaborar y los resultados mejores.
3. **Apoyan la documentación del código**: Aunque el código debe ser documentado, las pruebas son de ayuda a esta documentación, pues al estar bien estructuradas en casos de pruebas realizados sobre partes del programa, estos están bien documentados y da la medida de cómo utilizar y ayudan así al entendimiento del código probado.
4. **Separan la interfaz de la implementación**: Esto está dado ya que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
5. **Errores acotados y fáciles de localizar**: En este caso es sencillo desenmascarar un error pues al separar al software en partes, es más fácil encontrar un error en una unidad que en la totalidad del producto, y a su vez darle solución o tratamiento

Como se ha presentado anteriormente, las pruebas unitarias, tienen numerosas ventajas y a su vez tienen algunas características que son calificadas como “negativas o desventajas” aunque en la mayoría de los casos tienen su explicación lógica del porqué de su ocurrencia. A continuación se presentan una serie de problemáticas o desventajas de las pruebas de unidad:

1. No descubren todos los errores del código y esto se debe a que como su nombre lo indica, son pruebas que se efectúan sobre las unidades por si solas y no sobre el software de forma general.
2. No descubren errores de integración, problemas de rendimiento y otros problemas que afectan al producto de forma general.
3. Es difícil elaborar anticipadamente la lista de casos de pruebas con todas las combinaciones de posibles valores de entrada que puedan tomar las distintas variables de cada unidad que contiene un software.
4. Solo son efectivas si se usan en conjunto con otras pruebas de software como pueden ser las pruebas de integración, entre otras.

3.2 Diseño de las pruebas de unidad.

Para la ejecución de las pruebas de unidad se dispone de los siguientes recursos:

Físicos: ordenadores con CPU Core Duo 2.16 Ghz, memoria RAM de 1 Gb y un disco duro con capacidad de 100 Gbytes.

Lógicos: sistema operativo Debian GNU/Linux con núcleo 2.6.21, IDE de desarrollo Eclipse 3.2 con el plug-in CDT 2.1 C++, y framework para automatizar las pruebas CxxTest integrado al IDE y lenguaje utilizado.

Los casos de pruebas que han sido diseñados y ejecutados son del tipo Pruebas de Unidad y su objetivo fundamental es el de separar el código fuente que es probado y de esta forma validar su correcto funcionamiento método a método.

3.3 Pruebas a las clases dentro del escenario "Crear Reporte"

3.3.1 Clase TestTreeNodeReport

Prueba unitarias de la clase: **TreeNodeReport**

Casos de prueba del método: ***void setName (string name)***
Variables a considerar en el caso de prueba: ***string name***

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

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que cambie el valor del atributo name.	Válida

3.3.1.1 Casos de Prueba:

1- *void testSetName()*

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que se pueda cambiar el valor del atributo	Cadena pasada como parámetro.	La misma cadena pasada en los datos de entrada	La misma cadena pasada en los datos de entrada	

Casos de prueba del método: ***void setDocument (Document document)***
Variables a considerar en el caso de prueba: ***Document document***

Clase de Equivalencia para la variable: ***document***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que cambie el valor del atributo document.	Válida

3.3.1.2 Casos de Prueba:

2- *void testSetDocument()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

3.3.2 Clase TestShowProperties

Prueba unitarias de la clase: **ShowProperties**

Casos de prueba del método: ***doIt(ModelManager& modelManager)***
Variables a considerar en el caso de prueba: ***ModelManager& modelManager***

Clase de Equivalencia para la variable: ***modelManager***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Valores correctos para un objeto de tipo ModelManager.	Válida
2	Valores correctos para una objeto de tipo ModelManager.	Válida

3.3.2.1 3.3.2.1 Casos de Prueba:

1- *void testdoIt_1 ()*

2- *void testdoIt_2 ()*

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que el método funciona para un objeto de tipo ModelManager instanciado.	ModelManager modelManager = new ModelManager ();	Asignación correcta de la entrada a otra variable de tipo ModelManager;	Asignación correcta de la entrada a otra variable de tipo ModelManager ;	
2	Verificar que el método funciona para un objeto de tipo ModelManager no instanciado.	ModelManager modelManager = NULL;	Asignación correcta de la entrada a otra variable de tipo ModelManager;	Asignación correcta de la entrada a otra variable de tipo ModelManager ;	

Casos de prueba del método: ***void setInspectionableType(InspectionableType type)***

Variables a considerar en el caso de prueba: ***InspectionableType type***

Clase de Equivalencia para la variable: ***type***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que cambie el valor del atributo type.	Válida

3.3.2.2 Casos de Prueba:

3- *void testsetInspectionableType ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

Casos de prueba del método: ***void setTreeNodeReport(TreeNodeReport* treeNodeReport)***

Variables a considerar en el caso de prueba: ***TreeNodeReport* treeNodeReport***

Clase de Equivalencia para la variable: ***treeNodeReport***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que cambie el valor del atributo treeNodeReport.	Válida

3.3.2.3 Casos de Prueba:

4- *void testsetTreeNodeReport ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que se pueda cambiar	Objeto pasado como	El mismo objeto pasado	El mismo objeto pasado	

	el valor del atributo	parámetro.	en los datos de entrada	en los datos de entrada	

3.3.3 Clase TestPropertiesInspectorReport

Prueba unitarias de la clase: **PropertiesInspectorReport**

Casos de prueba del método: ***void setTreeNodeReport(TreeNodeReport* treeNodeReport)***

Variables a considerar en el caso de prueba: ***TreeNodeReport* treeNodeReport***

Clase de Equivalencia para la variable: ***treeNodeReport***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que cambie el valor del atributo treeNodeReport.	Válida

3.3.3.1 Casos de Prueba:

1- void testsetTreeNodeReport ()

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

Casos de prueba del método*: **void setProjectManager(ProjectManager& projectManager)***

Variables a considerar en el caso de prueba: ***ProjectManager& projectManager***

Clase de Equivalencia para la variable: ***projectManager***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que cambie el valor del atributo projectManager.	Válida

3.3.3.2 Casos de Prueba:

2- void testsetProjectManager ()

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

3.3.4 Clase TestPropertiesInspectorReportPage

Prueba unitarias de la clase: **PropertiesInspectorReportPage**

Casos de prueba del método: ***void setDocument (Document document)***

Variables a considerar en el caso de prueba: ***Document document***

Clase de Equivalencia para la variable: ***document***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que cambie el valor del atributo document.	Válida

3.3.4.1 Casos de Prueba:

1- void testSetDocument()

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

3.3.5 Clase TestPropertiesInspectorReportSection

Prueba unitarias de la clase: **PropertiesInspectorReportSection**

Casos de prueba del método: ***void setDocument (Document document)***

Variables a considerar en el caso de prueba: ***Document document***

Clase de Equivalencia para la variable: ***document***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que cambie el valor del atributo document.	Válida

3.3.5.1 Casos de Prueba:

1- void testSetDocument()

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

Casos de prueba del método: ***PropertiesReportObjectCollection save()***

Variables a considerar en el caso de prueba: ***PropertiesReportObjectCollection collection***

Clase de Equivalencia para la variable: ***collection***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que devuelve el valor correcto de la variable collection.	Válida

3.3.5.2 Casos de Prueba:

2- void testSave ()

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que devuelve el valor correcto de la colección creada.	collection = *none*	*none*	*none*	

3.4 Pruebas a las clases dentro del escenario "Insertar Componente"

3.4.1 Clase TestAddObject

Prueba unitarias de la clase: **AddObject**

Casos de prueba del método: ***doIt(ModelManager& modelManager)***

Variables a considerar en el caso de prueba: ***ModelManager& modelManager***

Clase de Equivalencia para la variable: ***modelManager***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Valores correctos para un objeto de tipo ModelManager.	Válida
2	Valores correctos para una objeto de tipo ModelManager.	Válida

3.4.1.1 Casos de Prueba:

1- *void testdoIt_1 ()*
2- *void testdoIt_2 ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que el método funciona para un objeto de tipo ModelManager instanciado.	ModelManager modelManager = new ModelManager ();	Asignación correcta de la entrada a otra variable de tipo ModelManager;	Asignación correcta de la entrada a otra variable de tipo ModelManager ;	
2	Verificar que el método funciona para un objeto de tipo ModelManager no instanciado.	ModelManager modelManager = NULL;	Asignación correcta de la entrada a otra variable de tipo ModelManager;	Asignación correcta de la entrada a otra variable de tipo ModelManager ;	

Casos de prueba del método: **void setGraphic(Graphic* graphic)**

Variables a considerar en el caso de prueba: ***Graphic* graphic***

Clase de Equivalencia para la variable: ***graphic***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que cambie el valor del	Válida

	atributo graphic.	

3.4.1.2 Casos de Prueba:

3- void testsetGraphic ()

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

Casos de prueba del método: ***void setX(WindowCoord x)***

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

Clase de Equivalencia para la variable: ***x***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	El valor de la variable estará entre los valores permisibles de la variable.	Válida

3.4.1.3 Casos de Prueba:

4- *void testSetX ()*

Caso de	Objetivo de la	Datos de	Salida	Salida	Observación

Prueba	Prueba	Entrada	Esperada	Obtenida	
1	Verifica que dado valores que se encuentren dentro de la frontera de validez se devuelvan los datos esperados	25	25	25	

Casos de prueba del método: ***void setY(WindowCoord y)***

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

Clase de Equivalencia para la variable: ***y***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	El valor de la variable estará entre los valores permisibles de la variable.	Válida

3.4.1.4 Casos de Prueba:

5- *void testSetY ()*

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	40	40	40	

3.4.2 Clase TestSection

Prueba unitarias de la clase: ***Section***

Casos de prueba del método: ***void 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 las Clases de Equivalencia**
1	El valor de la variable estará entre los valores permisibles de la variable.	Válida

3.4.2.1 Casos de Prueba:

1- *void testSetWidth ()*

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	75	75	75	

Casos de prueba del método: ***PrintFrequency getPrintFrecuency()***

Variables a considerar en el caso de prueba: ***PrintFrequency printFrecuency***

Clase de Equivalencia para la variable: ***printFrecuency***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que devuelve el valor correcto del atributo.	Válida

3.4.2.2 Casos de Prueba:

2- *void testGetPrintFrecuency ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que devuelve el valor correcto del atributo.	printFrecuency = *none*	*none*	*none*	

3.4.3 Clase TestGraphic

Prueba unitarias de la clase: **Graphic**

Casos de prueba del método: ***RGBAColor getColor()***

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 las Clases 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 ser 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 0 o por encima de 255.	Inválida

3.4.3.1 Casos de Prueba:

1- *void testGetColor_1 ()*
2- *void testGetColor_2 ()*

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.	RGBA color (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.	
2	Que se provoca alguna excepción al pasar un RGBAColor con valores no válidos.	RGBA color (-100,100,260,0)	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: ***void setColor(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 las Clases de Equivalencia
1	El valor de la variable estará entre los valores permisibles de la variable.	Válida

3.4.3.2 Casos de Prueba:

3- *void testSetColor ()*

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	0, 0, 0, 255	0, 0, 0, 255	0, 0, 0, 255	

Casos de prueba del método: ***BoundingRect getBoundaries()***

Variables a considerar en el caso de prueba: ***BoundingRect boundingRect***

Clase de Equivalencia para la variable: ***boundingRect***

Clase de	Clase de Equivalencia	Clasificación de las Clases de

Equivalencia		**Equivalencia**
1	Verificar que devuelve el valor correcto del atributo.	Válida

3.4.3.3 Casos de Prueba:

4- *void testGetBoundingRect ()*

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que devuelve el valor correcto del atributo.	*boundingRect = none*	*none*	*none*	

Casos de prueba del método: ***Graphic clone()***

Variables a considerar en el caso de prueba: ***Graphic graphicClone***

Clase de Equivalencia para la variable: ***graphic***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que devuelve el valor correcto de la variable graphicClone.	Válida

3.4.3.4 Casos de Prueba:

5- *void testClone ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que devuelve el valor correcto del objeto clonado.	*graphic = none*	*none*	*none*	

3.4.4 Clase TestSelectTool

Prueba unitarias de la clase: **SelectTool**

Casos de prueba del método: ***doIt(ModelManager& modelManager)***

Variables a considerar en el caso de prueba: ***ModelManager& modelManager***

Clase de Equivalencia para la variable: ***modelManager***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Valores correctos para un objeto de tipo ModelManager.	Válida
2	Valores correctos para una objeto de tipo ModelManager.	Válida

3.4.4.1 Casos de Prueba:

1- *void testdoIt_1 ()*
2- *void testdoIt_2 ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que el método funciona para un objeto de tipo ModelManager instanciado.	ModelManager modelManager = new ModelManager ();	Asignación correcta de la entrada a otra variable de tipo ModelManager;	Asignación correcta de la entrada a otra variable de tipo ModelManager ;	
2	Verificar que el método funciona para un objeto de tipo ModelManager no instanciado.	ModelManager modelManager = NULL;	Asignación correcta de la entrada a otra variable de tipo ModelManager;	Asignación correcta de la entrada a otra variable de tipo ModelManager ;	

Casos de prueba del método: ***void setSelectToolIndex(int index)***

Variables a considerar en el caso de prueba: ***int index***

Clase de Equivalencia para la variable: ***index***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que cambie el valor del atributo index.	Válida

3.4.4.2 Casos de Prueba:

3- *void testsetSelectToolIndex ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

Casos de prueba del método: **int *getSelectToolIndex ()***

Variables a considerar en el caso de prueba: ***int selectToolIndex***

Clase de Equivalencia para la variable: ***selectToolIndex***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que devuelve el valor correcto del atributo.	Válida

3.4.4.3 Casos de Prueba:

4- *void testGetSelectToolIndex ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que devuelve	*selectToolIndex = 1*	*1*	*1*	

	el valor correcto del atributo.				

3.4.5 Clase Test ReportEditorView

Prueba unitarias de la clase: **ReportEditorView**

Casos de prueba del método: ***void setDocument(Document& doc)***

Variables a considerar en el caso de prueba: ***Document& doc***

Clase de Equivalencia para la variable: ***doc***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que cambie el valor del atributo doc.	Válida

3.4.5.1 Casos de Prueba:

1- void testSetDocument ()

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que se pueda cambiar el valor del	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

	atributo				

Casos de prueba del método: ***Document getDocument ()***

Variables a considerar en el caso de prueba: ***Document document***

Clase de Equivalencia para la variable: ***document***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que devuelve el valor correcto del atributo.	Válida

3.4.5.2 Casos de Prueba:

2- void testGetDocument ()

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que devuelve el valor correcto del atributo.	*document = none*	*none*	*none*	

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

Variables a considerar en el caso de prueba: ***BoundaryBox boundaryBox***

Clase de Equivalencia para la variable: ***boundaryBox***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que devuelve el valor correcto del atributo.	Válida

3.4.5.3 Casos de Prueba:

3- *void testGetBoundaryBox ()*

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que devuelve el valor correcto del atributo.	*boundaryBoxt = none*	*none*	*none*	

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

Variables a considerar en el caso de prueba: ***SelectionRect selectionRect***

Clase de Equivalencia para la variable: ***selectionRect***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que devuelve el valor correcto del atributo.	Válida

3.4.5.4 Casos de Prueba:

4- *void testGetSelectionRect ()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que devuelve el valor correcto del atributo.	*selectionRect = none*	*none*	*none*	

3.5 Pruebas a las clases dentro del escenario "Mostrar Propiedades de los componentes"

3.5.1 Clase TestPropertiesInspectorReportGraphicObject

Prueba unitarias de la clase: **PropertiesInspectorReportGraphicObject**

Casos de prueba del método: ***void setGraphic (Graphic graphic)***

Variables a considerar en el caso de prueba: ***Graphic graphic***

Clase de Equivalencia para la variable: ***document***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia

1	Verificar que cambie el valor del atributo graphic.	Válida

3.5.1.1 Casos de Prueba:

1- void testSetGraphic()

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

Casos de prueba del método: ***void setDocument (Document document)***

Variables a considerar en el caso de prueba: ***Document document***

Clase de Equivalencia para la variable: ***document***

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que cambie el valor del atributo document.	Válida

3.4.5.5 Casos de Prueba:

2- void testSetDocument()

Caso de	**Objetivo de la**	**Datos de**	**Salida**	**Salida**	**Observació**

Prueba	Prueba	Entrada	Esperada	Obtenida	n
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

Casos de prueba del método: ***void setModelManager (ModelManager modelManager)***

Variables a considerar en el caso de prueba: ***ModelManager modelManager***

Clase de Equivalencia para la variable: ***modelManager***

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que cambie el valor del atributo modelManager.	Válida

3.4.5.6 Casos de Prueba:

3- void testSetModelManager ()

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que se pueda cambiar el valor del atributo	Objeto pasado como parámetro.	El mismo objeto pasado en los datos de entrada	El mismo objeto pasado en los datos de entrada	

Casos de prueba del método: **PropertiesCollection save()**

Variables a considerar en el caso de prueba: **PropertiesCollection collection**

Clase de Equivalencia para la variable: **collection**

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que devuelve el valor correcto de la variable collection.	Válida

3.5.1.2 Casos de Prueba:

3- void testSave ()

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que devuelve el valor correcto de la colección creada.	collection = *none*	*none*	*none*	

3.6 Conclusiones Parciales.

Por la complejidad del módulo fue necesario separar por escenarios las principales clases que podían ser probadas. Las pruebas de unidad confirmaron la existencia de algunos errores y validaron el buen comportamiento de las clases que no tenían problemas. Ayudaron a reutilizar componentes, que ya habían sido probados, en nuevas funcionalidades y con la seguridad de su correcto desempeño. Se

cumplió con el objetivo trazado pues quedó validada cada unidad probada y el código fuente de forma general.

Conclusiones

1. Después de valorar todos los diseñadores de reportes estudiados se llega a la conclusión de que todos son semejantes y ninguno llega a cumplir con las expectativas para su uso íntegro en un SCADA.
2. El paradigma de software libre facilita en gran medida la reutilización de código y un desarrollo acelerado de aplicaciones
3. El uso del patrón MVC y específicamente de la variante utilizada modelo-presentación, resulta muy conveniente para el desarrollo de un diseñador de informes para un SCADA, dado la facilidad de resistir el cambio de los requisitos y el alto desacoplamiento logrado con los demás subsistemas integrantes del SCADA.
4. La realización de la implementación basada en el análisis y diseño que permiten la interacción efectiva con el cliente y la validación de las funcionalidades exigidas en los requisitos.

Recomendaciones.

- Continuar el desarrollo de nuevas funcionalidades del diseñador de reportes del SCADA.
- Trabajar en el establecimiento de dependencias entre las clases genéricas planteadas en el diseño y las clases concretas pertenecientes a las bibliotecas GTK y Boost que faciliten a los desarrolladores la complementación del mismo y el posible cambio de tecnología en un futuro.
- Evaluar la posibilidad del uso del diseñador, de conjunto con el generador de reportes existente, en otros sistemas en el plano nacional utilizando las funcionalidades descritas en el presente trabajo.
- Estudiar el estado del arte de las herramientas similares de nueva generación con el objetivo de observar nuevas funcionalidades que puedan ser añadidas o actualizadas.

Referencias Bibliográficas.

1. JORGE, OSMANY. Modelación de un generador de informes para sistemas de supervisión, control y adquisición de datos. Ciudad de La Habana, Universidad de las Ciencias Informáticas, Mayo 2007
2. ZARAGOZA, BERNARDO y BACALLAO, RAUDI. Implementación de un módulo de generación de reportes para sistemas de supervisión, control y adquisición de datos. Ciudad de La Habana, Universidad de las Ciencias Informáticas, Julio 2008
3. PEREZ, YORJI. Implementación de componentes para el Editor Gráfico del SCADA. Ciudad de La Habana, Universidad de las Ciencias Informáticas, Julio 2008
4. SEHARA, YOSELL. Implementación de un modelo para la configuración de un sistema SCADA. Ciudad de La Habana, Universidad de las Ciencias Informáticas, Julio 2008

Bibliografía.

JAMES JACOBSON, I. B., GRADY Y RUMBAUGH. *El Proceso Unificado de Desarrollo de Software.* La Habana, 2004. p.

LARMAN, C. *UML y patrones. Introducción al análisis y diseño orientado a objetos*. 1. 2004. p.

---. *UML y patrones. Introducción al análisis y diseño orientado a objetos*. 2. 2004. p.

PRESSMAN, R. S. *Ingeniería de Software. Un enfoque práctico*. 5. La Habana, 2005. p.

BOOCH, G., JAMES JACOBSON, I. y RUMBAUGH, J. *UML Manual de referencia.* Addison Wesley. 1997.

Glosario de términos.

CSV(en inglés *comma-separated values*): Son un tipo de documento sencillo para representar datos en forma de tabla

Diseño de informe: Apariencia final con la cual será generada el informe.

Framework: En desarrollo de software, es una estructura en la cual pueden ser desarrollados distintos tipos de proyectos software. Normalmente incluye programas de ayuda, librerías de código y lenguajes de programación.

Generador de informes o reportes: Herramienta que permite generar reportes, también conocidos como informes a partir de datos primarios.

GIF (***G**raphics **I**nterchange **F**ormat*): Formato gráfico utilizado ampliamente en la World Wide Web, tanto para imágenes como para animaciones. Es un formato sin pérdida de calidad, siempre que partamos de imágenes de 256 colores o menos. Una imagen de alta calidad, como una imagen de color verdadero (profundidad de color de 24 bits o superior) debería reducir literalmente el número de colores mostrados para adaptarla a este formato, y por lo tanto existiría una pérdida de calidad. GIF llegó a ser muy popular porque podía usar el algoritmo de compresión LZW (Lempel Ziv Welch) para realizar la compresión de la imagen, que era más eficiente que el algoritmo Run-Lenght Encoding (RLE) que usaban formatos como PCX y MacPaint. Por lo tanto, imágenes de gran tamaño podían ser descargadas en un razonable periodo de tiempo, incluso con módems muy lentos.

GNOME: GNU **N**etwork **O**bject **M**odel **E**nvironment es un entorno de escritorio basado en las librerías GTK diseñadas para GIMP para sistemas operativos de tipo Unix bajo tecnología X Window. Al igual que KDE, permite la interacción entre programas. Se encuentra disponible actualmente en más de 35 idiomas. Forma parte oficial del proyecto GNU.

GPL: Son las siglas de General Public License, Licencia Pública General, definida por la Fundación para el Software Libre (FSF) para proteger los derechos de copia del software libre.

GUI (Graphical User Interfaces): Componente de una aplicación informática que el usuario visualiza y a través de la cual opera con ella. Está formada por ventanas, botones, menús e iconos, entre otros elementos.

HTML (***H****yper****T****ext* ***M****arkup* ***L****anguage*, lenguaje de marcas hipertextuales): Lenguaje de marcación diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web.

Informe o reporte: Documento contentivo de información personalizada y útil para el destinatario del mismo.

KDE (**K D**esktop **E**nvironment): Es un entorno de escritorio gráfico e infraestructura de desarrollo para sistemas Unix y, en particular, GNU/Linux.

LGPL: Son las siglas de Lesser General Public License o Library General Public License. Esta licencia permite el enlace dinámico de aplicaciones libres a aplicaciones no libres.

Multiplataforma: Es un término utilizado frecuentemente en informática para indicar la capacidad o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

Framework: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

PDF (*P**ortable*** ***D****ocument* ***F****ormat*, Formato de Documento Portátil): Formato de almacenamiento de documentos multiplataforma (Microsoft Windows, Unix, Mac) desarrollado por la empresa Adobe System. Especialmente ideado para documentos susceptibles de ser impresos.

Plataforma: Base, elemento de apoyo

XML (*e****X****tensible* ***M****arkup* ***L****anguage*, 'lenguaje de marcas extensible'): Metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Permite definir la gramática de lenguajes específicos, por lo tanto XML no es realmente un lenguaje en particular, sino una manera

de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG y MathML.