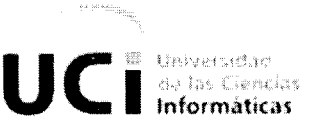
Universidad de las Ciencias Informáticas
Facultad 5



Título: Implementación de componentes para el Editor Gráfico de Reporte del SCADA

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

Autor: Yorji Pérez Hernández

Tutor(es): MSc. Jaime Fardales Pérez.
Ing. Amado Espinosa Hidalgo.

Julio, 2008

Agradecimientos

A todas las personas que me ayudaron y guiaron en la realización de este trabajo: Molina, Enrique, Kenia, Raudy, Bernardo, Iliana, mi tutor Fardales, y demás.

A la familia SCADA, que me ha ido enseñando a ser cada día más profesional.

A mis padres por ser siempre tan dedicados y preocupados por mis estudios y darme su voto de apoyo en mis decisiones.

A mi novia por su comprensión.

A mis compañeros de Joven Club que me enseñaron a dar los primeros pasos en el mundo de la informática.

A mis abuelos y familia en general por sus consejos.

A mis amigos todos, se les quiere, y de corazón gracias por estar, gracias dos veces por existir, y si me dieran a escoger diría gracias por ponerlos en mi camino.

Dedicatoria

A mis maravillosos padres los Pérez Pones por todo su amor y dedicación, por enseñarme y guiarme, por siempre estar, por ser así de lindos y grandes sus corazones, por abrazar tan fuerte, por enseñarme a mí y mi hermano lo que es el calor de una familia.

A mi hermano yoelito, por su cariño.

A mi abuela Milagros por todos sus recuerdos presentes y estar siempre dentro de mí.

A mis amigos todos que son mi familia.

Resumen

Un Sistema de Adquisición de Datos y Control Supervisorio (SCADA) en inglés, "Supervisory Control And Data Adquisition" es una aplicación de software especialmente diseñada para funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo y controlando el proceso de forma automática desde la pantalla del operador. Además, provee a diversos usuarios, toda la información que se genera en el proceso productivo, control de calidad, supervisión, mantenimiento. (SCADA 2005)

Entre los módulos o bloques de software que desarrolla un SCADA se encuentra el Módulo de Reportes, que se encarga de generar reportes y diseñar las plantillas utilizadas para la generación de los mismos.

Un diseñador de plantillas utiliza diferentes componentes para el desarollo de éstas, de ahí la necesidad de desarrollar componentes complejos y de suma importancia para el acabado de la herramienta, objetivo que pretende alcanzar la presente investigación, para de esta manera hacer más interactiva y funcional la edición de los reportes.

Palabras Claves: Componente, Diseño, Plantilla, Reporte, Secciones, SCADA.

Índice

Índice de Figuras

Índice de Tablas

Introducción

La automatización de los procesos industriales es uno de los mecanismos que permiten el desarrollo de la sociedad, en búsqueda del aumento en los niveles de eficiencia, disminución de costos y optimización de los procesos.

Hoy los sistemas SCADA son un eslabón importante y estratégico para todas aquellas naciones que pretendan industrializarse. Buscando la independencia tecnológica, surge en el 2006 el proyecto SCADA cuyo objetivo es crear un sistema que supervise, controle, optimice y gestione los procesos industriales.

A partir de la importancia que representa para el país el desarrollo del proyecto SCADA, la Universidad de Ciencias Informáticas (UCI) y de forma particular dentro de ésta el Polo de Hardware y Automática, asume la responsabilidad de gestionar el proyecto, conformando un equipo multidisciplinario de estudiantes, profesores y especialistas en áreas de automatización de todo el país para el impulso de esta importante tarea, que exigía el desarrollo bajo paradigmas y plataformas de software libre.

Un sistema SCADA está compuesto por diferentes módulos o bloques de trabajo, algunos de ellos son los de configuración, módulo de procesos, gestión de archivos y datos, comunicación, interfaz gráfica con el operador y reportes. Este último se compone de dos fases esenciales: la generación de reportes y el diseño de las plantillas usadas para la generación de los mismos. Estas fases permiten a operadores y mantenedores interactuar con el sistema y obtener de una forma más fácil la información registrada en él.

La meta de elaborar un sistema de generación de reportes al nivel del estado del arte como complemento al sistema SCADA que se desarrolla en la UCI, se enfrentó dividiendo la construcción entre dos equipos de trabajo. Un primer equipo que tiene como responsabilidad la fase de generación de los reportes, y el segundo asume el módulo que permite la edición de las plantillas. Debido a la magnitud del desarrollo es necesario distribuir a su vez las tareas para la confección del editor de plantillas a partir de los componentes fundamentales identificables. Entre éstos podemos mencionar los gráficos, campos calculados, campo de datos, entre otros.

También es necesario destacar que existen un conjunto de elementos coyunturales que condicionan el proceso de elaboración de los componentes antes referidos. Entre los que destacan: la reutilización de bibliotecas y tecnologías en software libre disponibles o

seleccionadas por la dirección del proyecto que contribuyan a garantizar la calidad del resultado final así como permiten acortar los tiempos de desarrollo.

A partir de los elementos anteriormente presentados como **situación problémica** se propone el siguiente **problema**: ¿Cómo elaborar los componentes de manejo campos calculados, gráficos y campos de datos utilizando las tecnologías seleccionadas en el proyecto SCADA de forma eficiente?

Se identifican a los sistemas de edición visuales de plantillas para generadores de reportes como **objeto de estudio** y de forma más específica la implementación interna de las funcionalidades como **campo de acción.**

De acuerdo con el problema científico planteado la **idea a defender** es la siguiente:
Si se desarrollan los componentes campos calculados, gráficos y campos de datos utilizando bibliotecas y tecnologías de software libre previamente seleccionadas, así como desarrollando su implementación eficiente se podrán integrar dichos componentes en una aplicación de edición de plantillas que tendrán un nivel de calidad acorde a los requisitos.

El **objetivo general** planteado en este trabajo es implementar los componentes **campos calculados, gráficos y campos de datos para** el diseñador de reportes sobre la plataforma de software libre.

Para dar cumplimiento al objetivo general se plantearon las siguientes **tareas de investigación** que rigieron el desarrollo de este trabajo, las cuales se enuncian a continuación:

- Recopilar y evaluar información sobre diferentes diseñadores de reportes existentes para plataformas en software libre.
- Análisis de las distintas herramientas con que cuentan otros diseñadores para así hacer un estudio de estado del arte sobre sus funcionalidades.
- Diseñar las clases correspondientes a los componentes campo calculado, campo de datos y gráficos.
- Implementar cada una de las soluciones para los componentes anteriormente mencionados.
- Validar la solución propuesta mediante la realización de pruebas de unidad.

Esperándose como posibles resultados:

- Obtener con la implementación de los componentes campo calculado, campo de datos y gráficos parte importante de la paleta de diseño del diseñador de reportes.

- Con el diseño y desarrollo de las pruebas se espera que cada uno de los componentes implementados para la herramienta cumpla con los requerimientos impuestos al módulo, además de un código con la calidad requerida.

El presente documento está estructurado en tres capítulos:

En el **Capítulo 1** se describe la concepción general de un sistema SCADA, se hace referencia a las características que debe cumplir un editor de reportes, se hace un estudio del arte sobre los principales diseñadores de reportes en software libre y se canalizan conceptos que ayuden a una correcta implementación de los componentes más complejos. Además se describen cada una de las herramientas o tecnologías utilizadas para el desarrollo de los componentes tratados en este trabajo.

En el **Capítulo 2** se evalúan los parámetros para una correcta implementación de los componentes, para ello se describen las principales funcionalidades que debe cumplir cada uno de los mismos, centrándose el trabajo en las diferentes propiedades que debe manejar cada uno de ellos. Se describen las clases y algunos de los métodos más importantes. En este capítulo también se explica la lógica para el correcto funcionamiento de los componentes de manejo: campos calculados, gráficos y campo de datos.

En el **Capítulo 3** Se presentan pruebas realizadas al sistema en busca de debilidades en el código con el objetivo de corregirlas.

Capítulo 1: Fundamentación Teórica

Introducción

En el siguiente capítulo se describe la concepción general de un sistema SCADA, se hace referencia a las características que debe cumplir un editor de reportes, se hace un estudio del arte sobre los principales diseñadores de reportes en software libre y se canalizan conceptos que ayuden a una correcta implementación de los componentes más complejos. Además se describen cada una de las herramientas o tecnologías utilizadas para el desarrollo de los componentes tratados en este trabajo.

1.1 Sistema SCADA y Generación de Reportes

Se le da el nombre SCADA (*Supervisory Control And Data Acquisition* o Control con Supervisión y Adquisición de Datos) a cualquier software que permite el acceso a datos remotos de un proceso y posibilite utilizando las herramientas de comunicación necesarias en cada caso, el control del mismo. Atendiendo a la definición no se trata de un sistema de control, sino de una utilidad software de motorización o supervisión, que realiza la tarea de interfaz entre los niveles de control (PLC o Controladores Lógicos Programables) y los de gestión superior.

Los sistemas SCADA se conciben principalmente como una herramienta de supervisión y mando. Cuando se habla de un sistema como éste, no hay que olvidar que hay algo más que las pantallas que nos informan de cómo van las cosas en la instalación. Tras éstas se encuentran múltiples utilidades de software que pretenden que el sistema funcione de forma eficiente y segura.

Gracias al SCADA se consigue una localización más rápida de errores, se minimizan los períodos de paro en las instalaciones y repercute en la reducción de costes de mantenimiento. Los programas de visualización pueden presentar todo tipo de ayuda al usuario, desde la aparición de una alarma, hasta informes completos del estado de la industria.

Es cada vez más común la tendencia a complementar las funcionalidades de adquisición y registro de datos con la capacidad de generar información capaz de ayudar en la toma de decisiones. Es por ello que SCADA cuenta con un módulo de reportes, creado para hacer garantizar que se logre una herramienta capaz de generar información referente a la situación

de la industria o lugar donde se instale el SCADA. El generador de reportes es la herramienta que permite al SCADA emitir toda esa información.

Una de las vías posibles para hacer emitir los reportes puede ser el uso de aplicaciones que permitan la personalización de la información extraída de la base de datos del SCADA, se puede mencionar como una de estas herramientas el Excel. Otra vía es la de definir un conjunto de reportes prediseñados anteriormente, ésta llevaría como inconveniente a que el usuario sólo pudiera usar dichos reportes ya predefinidos por los desarrolladores del SCADA, y por supuesto la vía más factible y viable para los usuarios sería la de diseñar sus propios reportes o informes mediante el uso de un diseñador de reportes.

Por todo lo antes descrito se hace necesario asumir la tarea de realizar la herramienta del diseñador de reportes, cuya importancia para que los usuarios que usen el sistema SCADA puedan definir los reportes por ellos mismos.

Es importante no dejar de mencionar 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.

1.2 Principales diseñadores de reportes en Software Libre

A continuación se mostrarán algunos ejemplos de distintas herramientas que contienen diseñadores de plantillas o reportes. De los mismos se hizo un estudio del estado del arte, siendo más específicos en su paleta de componentes y dentro de ésta los que en este trabajo se trata, para así ganar en experiencia en este aspecto.

1.2.1 NCReport

NCReport es una herramienta de diseño y generación de informes que permite diseñar de forma visual los informes además de posibilitar su impresión. El ambiente de trabajo para el diseño de informes es bastante amigable al facilitar el acceso a los componentes necesarios para el diseño mediante una barra de componentes y a través del menú. Los componentes que se pueden utilizar son: fecha y hora, campo de expresión, tabla de referencia cruzada, número de página, gráfico, texto fijo o etiquetas, imágenes estáticas e hipervínculos. (NCREPORT).

Los diseños de informes son almacenados en formato XML. Está desarrollado usando C++ basado en un toolkit de QT, razón por la cual cualquier programador puede comunicarse con él directamente reutilizando sus clases. (Ver Anexo 1)

1.2.2 Kugar

Kugar 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. (KUGARREPORTS)

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. Ésto trae consigo toda una gama de plantillas para a partir de ellas empezar a trabajar. (Ver Anexo 2)

Para la realización de los distintos tipos de diseños el Kugar cuenta con:

- *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.

1.2.3 JasperReports

JasperReports (JASPERREPORTS) 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. Puede usarse en aplicaciones de Java, incluyendo J2EE (plataforma JAVA2 edición empresarial) o aplicaciones Web, para generar contenidos dinámicos.

Esta librería puede usarse en GNU/Linux y en Microsoft Windows, y se distribuye bajo licencia LGPL. (Ver Anexo 3)

1.2.4 Rekall

Rekall(REKALLREVEALED.ORG) es una interfaz de usuario para bases de datos (Front-End) en entornos KDE similar a Microsoft Access. Sin embargo, no es y no incluye una base de datos. Debido a ésto, los datos están almacenados en otra parte como en un servidor SQL. De esta forma Rekall permite gestionar de una forma totalmente gráfica bases de datos (por ahora MySQL, PostgreSQL, xBase con XBSQL, IBM DB2 y ODBC), permitiendo el diseño de formularios e informes, creación de 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 GNU/Linux.

Rekall también puede diseñar y usar formularios e informes, construir consultas a base de datos, importar y exportar datos. (Ver Anexo 4)

Rekall está escrito en C++ / QT, aunque en la página oficial de su creador se aclara que no solo se trabaja en versiones soportadas sobre QT, para hacerlo totalmente independiente de las bibliotecas de KDE. Actualmente está disponible para GNU/Linux y Microsoft Windows y se está trabajando en una versión para Mac OS.

1.2.5 DataVision

DataVision es una herramienta de informes para bases de datos similar a Crystal Reports. Los informes pueden ser diseñados usando una interfaz gráfica. Es un producto que disfruta de muchas de las características avanzadas de 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 programado en Java y corre en diversas 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.

1.3 ¿Qué es un diseño de reporte?

Un diseño de reporte no es más que una plantilla o modelo que sirve para establecer la estructura final de los reportes que se generarán basados en su uso.

La apariencia visual de un diseño está formada por una serie de componentes colocados sobre distintas secciones bien delimitadas las cuales definirán el comportamiento a la hora de generación de los componentes en ellas contenidos, sin embargo con estos componentes visuales no es suficiente para definir todas las características de un diseño, razón por la cual será necesario especificar una serie de propiedades y mecanismos que permitan crear diseños verdaderamente funcionales para el SCADA.

1.4 Diseñador de Reportes para el SCADA

En el SCADA el diseñador de reportes está contenido dentro del módulo de Reportes, dicha herramienta permite diseñar las plantillas a partir de las cuales se puedan generar nuevos reportes, ésta permite personalizar los diseños incorporando conceptos de formato como los encabezados, pie y número de página. Posee funcionalidades para la sumarización de los datos que permiten el cálculo de varianzas, medias y totalización así como componentes para la visualización de gráficos de barra, líneas, puntos y pastel.

El diseñador de reportes es la herramienta que permite al mantenedor gestionar los diseños de reportes que son la base para la generación de los distintos reportes del SCADA. Mediante el uso de esta herramienta el mantenedor puede crear nuevos diseños o modificar los ya existentes.

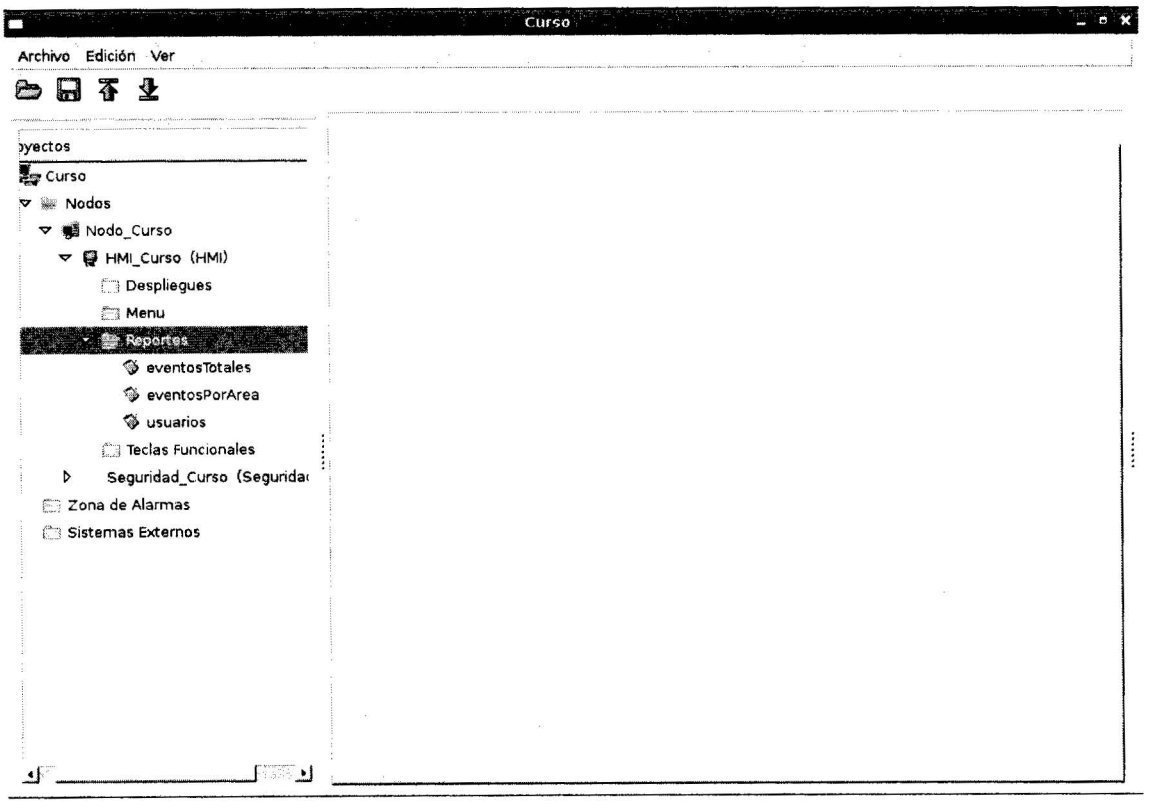


Fig. 1 Creación de diseños

Después de completar una serie de datos generales del diseño se desplegará el ambiente de edición (diseñador) que se usará para diseñar las plantillas o diseños de reportes. Como se muestra en la Fig. 2, este diseñador cuenta con una área central donde se desarrollará toda la actividad de diseño, o sea donde se irá construyendo el diseño deseado. Para la construcción de los mismos, se dispondrá de una paleta de componentes visuales, los cuales podrán ser insertados sobre las distintas secciones del área de diseño, esta paleta de componentes se despliega en el extremo derecho de la ventana.

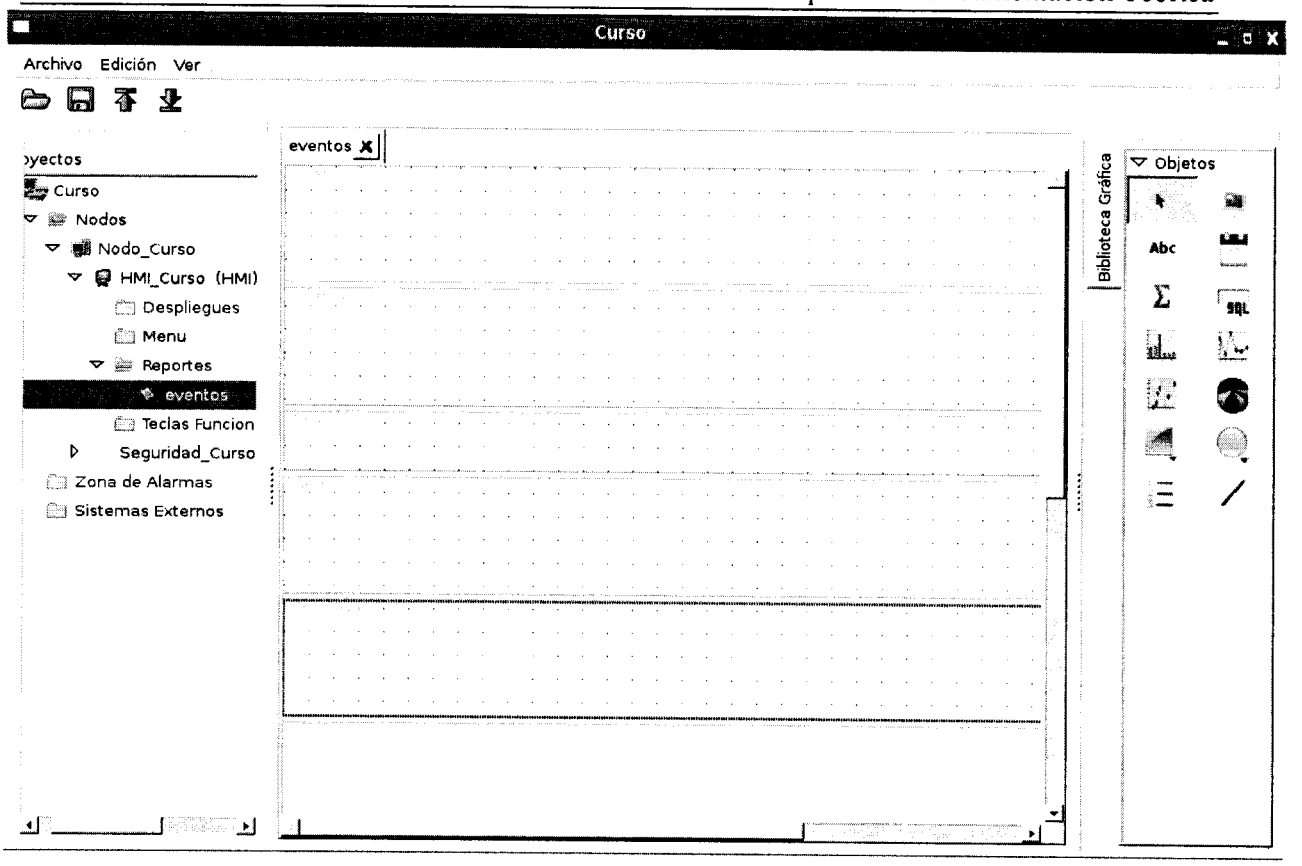


Fig. 2 Diseñador de Reportes

1.4.1 Secciones

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, como se podrá observar en la figura anterior el área de diseño aparece dividido en 5 fragmentos verticales divididos por líneas dobles de color gris, éstas son las referidas secciones. Existen varios tipos de secciones cuya peculiaridad fundamental es su comportamiento en momento de generación, en la Fig. 3 se muestra de forma esquemática las principales secciones que pueden formar parte de un diseño y que por demás son las que aparecerán por defecto al crear un nuevo diseño.

Encabezado de reporte
Encabezado de página
Detalles

Pie de reporte
Pie de página

Fig. 3 Secciones básicas de un diseño

En este esquema se puede apreciar la división del diseño en 5 secciones, el uso típico de estas secciones es como sigue:

- 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 cada vez que sea necesario.
- 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.

1.4.2 Componentes visuales para un diseño

Como se refirió anteriormente, para la elaboración de diseños se dispone de una paleta de componentes o herramientas gráficas. 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.

1.5 Componentes definidos para el diseñador de Reportes del SCADA

Están disponibles para la realización de diseños los siguientes componentes:

- ***Campo de tiempo:*** Etiqueta encargada de colocar en los reportes la fecha, pone la hora actual, día, mes y año.
- ***Campo calculado:*** Permite incluir en el informe cuentas, sumas, cálculos de varianza, media ponderada, máximos, mínimos, etc.
- ***Campo de Datos:*** Este componente es el encargado de mostrar los datos que se almacenan en la base de datos.
- ***Etiqueta:*** Un área rectangular que puede tener fronteras y se puede llenar por cualquier clase de datos textuales.
- ***Rectángulo***: Permite refinar el aspecto final del informe insertando rectángulos.
- ***Circulo***: Permite refinar el aspecto final del informe insertando círculos.
- ***Líneas:*** Permiten refinar el aspecto final del informe insertando líneas.
- ***Gráficos:*** Se incluyen distintos tipos de gráficos tales como gráficos de pastel, gráficos de puntos, líneas y barras.
- ***Imágenes:*** Permite incluir al reporte las imágenes.
- ***Número de Página***: Representa el número de página a medida que se va generando el reporte, va enumerando las páginas.

Este trabajo se va a referir principalmente, a darle solución a los componentes campo calculado, campo de datos y gráficos.

1.6 Herramientas y entornos de desarrollo utilizados

A continuación se describen cada una de las bibliotecas utilizadas para el desarrollo de este trabajo, además se hace una breve descripción del lenguaje de programación a utilizar así como de la biblioteca matemática a emplear por uno de los componentes de los que se tratará en este trabajo. Además se hace una descripción del sistema operativo.

Es por ello y en vista a que la Universidad de Ciencias Informáticas (UCI) ha ido migrando parcialmente al sistema operativo **GNU/Linux,** y se dispone a llevar sus productos para la plataforma haciendo uso de las herramientas que brinda el software libre, se da la opción de implementar el SCADA apoyado en **Linux**, puesto que sería una manera de reducir los costos, además de tener la libertad de uso y distribución de los programas sin incurrir en litigios de licenciamiento o asuntos legales.

1.6.1 GNU/Linux

Una distribución de Linux es una variante de ese sistema operativo que incorpora determinados paquetes de software para satisfacer las necesidades de un grupo específico de usuarios, dando así origen a ediciones hogareñas, empresariales y para servidores. Pueden ser exclusivamente de software libre, o también incorporar aplicaciones o controladores propietarios.

El Proyecto Debian es una asociación de personas que han hecho causa común para crear un sistema operativo (SO) libre, bajo la licencia General Public Licence (LGPL). Basado en el conocido y distribuido núcleo de Linux la distribución es llamada Debian.

Una gran parte de las herramientas básicas que completan el sistema operativo, vienen del proyecto GNU; de ahí el nombre: GNU/Linux.

Debian es la única distribución importante de GNU/Linux mantenida solamente por voluntarios, lo que permite la actualización de los paquetes de software y la participación abierta de todos aquellos que deseen colaborar.

Se decidió usar la distribución Debian porque es una distribución de GNU/Linux de desarrollo muy estable, por lo que los paquetes que se desarrollen en él y quieran ejecutarse utilizando cualquier distribución siempre serán estables. A diferencia de otras distribuciones tiene un magnífico soporte de estabilidad en las aplicaciones (no requieren ser compiladas en la máquina que las esté usando). Los módulos del LDAP (Lighweight Directory Access Protocol) se pueden ejecutar sin problemas permitiendo que los usuarios usen sus sesiones en cualquier máquina dentro del área de trabajo, ahorrando recursos de hardware.

1.6.2 Cairo

Cairo es una biblioteca de gráficos, soporta imágenes de composición que pueden ser utilizadas con GDK, gráficos vectoriales, entre otros, es un API además de GDK/GTK (en lugar de un sustituto). Las funciones de este módulo proporcionan los medios de dibujo, las operaciones de cairo incluyen rellenos, pinceladas, curvas Bezier, todas las operaciones de dibujo se pueden lograr mediante las transformaciones básicas (rotar, escalar, mover). Es una biblioteca completamente libre con soporte para múltiples dispositivos de salida, entre ellos Win32, PostScript, PDF, archivos SVG entre otros.

1.6.3 GTK+

GTK+ es un grupo importante de bibliotecas o rutinas para desarrollar interfaces gráficas de usuario para principalmente los entornos gráficos GNOME, XFCE Y ROX de sistemas Linux. GTK+ es la abreviatura de GIMP toolkit (conjunto de rutinas para GIMP). Es software libre (bajo la licencia LGPL), multiplataforma y parte importante del proyecto GNU.

GTK+ se basa en varias bibliotecas del equipo de GTK+ y de GNOME entre ellas podemos mencionar a la biblioteca de renderizado avanzado de controles de aplicación Cairo. Actualmente es muy usada por muchos programas en los sistemas GNU/Linux. GTK+ se ha diseñado para permitir programar con lenguajes como C, C++, C#, Java, Perl, PHP o Python.

1.6.4 IDE Eclipse

El Eclipse es un Entorno Integrado de Desarrollo (IDE) multiplataforma libre para crear aplicaciones clientes de cualquier tipo. La primera y más importante aplicación que ha sido realizada con este entorno es el afamado IDE Java llamado Java Development Toolkit (JDT) y el compilador incluido en Eclipse, que se usaron para desarrollar el propio Eclipse.

El IDE de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. El mecanismo de módulos permite que el entorno de desarrollo soporte otros lenguajes además de Java. Por ejemplo, existe un módulo para dar soporte a C/C++. Existen módulos para añadir un poco de todo, desde Telnet hasta soporte a bases de datos.

1.6.5 C++

El C++ es un lenguaje de programación diseñado a mediados de los años 1980. Está considerado por muchos como uno de los lenguajes más potentes, debido que permite trabajar tanto a alto nivel como a bajo nivel. Se puede decir que C++ abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos, dicho lenguaje gana mucho en potencia ya que tiene la posibilidad de sobrecargar operadores. Las principales características del C++ son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica.

1.6.6 Biblioteca GSL

La biblioteca científica de GNU (GSL) es una biblioteca numérica para C y C++, es software libre bajo la licencia pública general de GNU. La biblioteca ofrece una amplia gama de rutinas matemáticas, tales como cálculo de máximos, mínimos, varianzas entre otras, existen más de mil funciones en general con un extenso conjunto de pruebas. GSL incluye una referencia de 500 páginas de manual en formato texto.

La biblioteca GSL se le da soporte o se desarrolla en la plataforma GNU/LINUX pero además se sabe que compila en las siguientes plataformas:

- SunOS 4.1.3 y Solaris 2.x (Sparc).
- Alpha GNU / Linux, gcc.
- HP-UX 9/10/11, PA-RISC, gcc / cc.
- IRIX 6,5, gcc.
- M68k NeXTSTEP, gcc.
- Alpha de Compaq Tru64 Unix, gcc.
- FreeBSD, NetBSD y OpenBSD, gcc.
- Cygwin.
- Apple Darwin 5,4.
- Super Hitachi SR8000 Técnica Server, cc.

Es importante mencionar que dicha biblioteca está libre para el uso de cualquier usuario y en caso de necesitarlo se puede modificar el código fuente a conveniencia o necesidad del usuario que la use.

1.6.7 CxxTest

Es una biblioteca utilizada para el desarrollo de pruebas. Está diseñada para ser tan portátil como sea posible. De fácil uso algunas de sus ventajas son:

- No requiere de funciones miembro de plantilla.
- No requiere de manejo de excepciones.
- No requiere ninguna biblioteca externa.

Se distribuye en su totalidad como un conjunto de archivos de cabecera que le hace extremadamente portátil y utilizable.

1.7 Metodología de desarrollo de software

La metodología de desarrollo de software a utilizar es RUP definida en los inicios del proyecto SCADA. A continuación se hace una descripción de la metodología y del lenguage de modelado utilizado para modelar los componentes sobre la herramienta *Rational Software Architect* (RSA) .

1.7.1 Metodología RUP

El Proceso Unificado de Desarrollo (RUP), en inglés, Rational Unified Process, fue desarrollado en 1998 por Grady Booch, Ivar Jacobson y James Rumbaugh, prominentes metodologistas en la industria de la tecnología y sistemas de información. RUP se caracteriza por ser: dirigido por Casos de Uso, centrado en la arquitectura, iterativo e incremental.

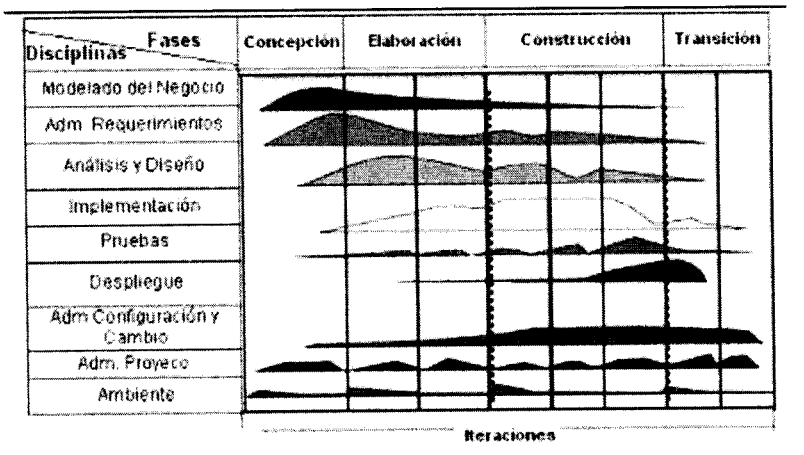


Fig. 4 Metodología de desarrollo de software: RUP

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. En la Fig.4 se muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una línea base de la arquitectura.

Durante la fase de inicio las iteraciones hacen ponen mayor énfasis en actividades modelado del negocio y de requerimientos.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

Como se puede observar en cada fase participan todas las disciplinas, pero que dependiendo de la fase el esfuerzo dedicado a una disciplina varía.

1.8 UML 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

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas, a continuación se mencionan los mismos:

Los **Diagramas de Estructura** enfatizan en los elementos que deben existir en el sistema modelado:

- Diagrama de clases
- Diagrama de componentes
- Diagrama de objetos
- Diagrama de estructura compuesta (UML 2.0)
- Diagrama de despliegue
- Diagrama de paquetes

Los **Diagramas de Comportamiento** enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de actividades
- Diagrama de casos de uso
- Diagrama de estados

Los **Diagramas de Interacción** son un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de secuencia

- Diagrama de colaboración
- Diagrama de tiempos (UML 2.0)
- Diagrama de vista de interacción (UML 2.0)

Capitulo 2: Descripción de la solución.

Introducción

En el presente capítulo se evalúan los parámetros para una correcta implementación de los componentes, para ello se describen las principales funcionalidades que debe cumplir cada uno de los mismos, centrándose el trabajo en las diferentes propiedades que debe manejar cada uno de ellos. Se describen las clases y algunos de los métodos más importantes. En este capítulo también se explica la lógica para el correcto funcionamiento de los componentes de manejo: campos calculados, gráficos y campo de datos.

2.1 Funcionalidades y propiedades de los componentes

2.1.1 Campo de datos

Este componente es el usado para el despliegue visual del resultado de una operación. En este caso las operaciones estarán asociadas al manejo de volúmenes de datos provenientes de bases de datos. Por lo cual se deberán establecer las siguientes propiedades para definirlo:

- Nombre: Nombre que identificará el componente en el diseño.
- Consulta: De la ejecución de esta consulta se obtendrán los datos para nutrir el campo de datos.
- Columna: Con esta propiedad se especifica la columna a desplegar en el campo de datos correspondiente al flujo de datos que provoca la ejecución de la consulta.
- Posición X: Coordenada en el eje horizontal que se corresponde con el extremo superior izquierdo del rectángulo que enmarca el campo, todos los valores a desplegar se desplegarán en esta misma coordenada.
- Posición Y: Coordenada en el eje vertical que se corresponde con el extremo superior izquierdo del rectángulo que enmarca el campo, el conjunto de valores que nutrirán este campo se desplegarán en el reporte de forma equidistante a partir de esta posición.
- Largo: Extensión horizontal del rectángulo previsto para desplegar los valores.
- Ancho: Extensión vertical del rectángulo para desplegar los valores.
- 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: Color que tomará el texto.

Fig. 5 Interfaz para establecer las propiedades del Campo de datos

2.1.2 Campo calculado

En un reporte se pueden realizar una serie de cálculos para lo cual se emplearán variables cuya especificación se explica más adelante en este documento. Este componente se usará para mostrar en un reporte el valor almacenado por una variable.

- Nombre: Nombre que identificará el componente en el diseño.
- Variable: Variable de donde se obtendrá el valor que aparecerá en el reporte.
- Posición X: Coordenada en el eje horizontal que se corresponde con el extremo superior izquierdo del rectángulo que enmarca el campo.
- Posición Y: Coordenada en el eje vertical que se corresponde con el extremo superior izquierdo del rectángulo que enmarca el campo.
- Largo: Extensión horizontal del rectángulo que enmarca el campo.
- Ancho: Extensión vertical del rectángulo que enmarca el campo.
- 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: Color que tomará el texto.

Por supuesto en el momento de ser generado el reporte, el rectángulo empleado para representar el campo calculado en tiempo de diseño, será sustituido por el valor que almacena la variable establecida como parámetro de este componente.

Fig. 6 Interfaz para establecer las propiedades del Campo calculado

2.1.3 Gráficos

La inserción de gráficos en los reportes es una de las formas más amenas y didácticas de presentación de información. En los reportes del SCADA se pueden incluir gráficos que muestren la información procedente de flujos de datos en diferentes formas. En esta versión del generador de reportes se podrán incluir gráficos de barra, líneas, puntos y de pastel. Las propiedades que deberán ser establecidas para componentes de este tipo son las siguientes:

- Nombre: Nombre que identificará el gráfico en el diseño.
- Título: Mediante este atributo se establece el título que aparecerá para describir el gráfico generado.
- Posición X: Coordenada en el eje horizontal que se corresponde con el extremo superior izquierdo del rectángulo que enmarca el gráfico.
- Posición Y: Coordenada en el eje vertical que se corresponde con el extremo superior izquierdo del rectángulo que enmarca el gráfico.
- Largo: Extensión horizontal del rectángulo que enmarca el gráfico.
- Ancho: Extensión vertical del rectángulo que enmarca el gráfico.

- Color del fondo: Color que se le asignará al fondo del gráfico.

Para el caso de los gráficos de tipo barras y puntos, y de líneas se deben definir, además de las anteriores propiedades, las siguientes:

- Color de los ejes: Color que tomarán los ejes horizontal y vertical del gráfico.
- Tipo: Mediante este campo se especifica el tipo de gráfico que se generará con el flujo de datos (Barras o punto).
- Etiqueta asociada al eje horizontal: Texto que aparecerá para describir los valores que se representan en el eje horizontal.
- Consulta asociada al eje horizontal: De la ejecución de esta consulta se obtendrán los datos que conformarán la ordenada horizontal de cada punto a representar.
- Columna asociada al eje horizontal: Con esta propiedad se especifica la columna a ser usada como vector de ordenadas en el eje horizontal. Para que se obtengan pares ordenados que se puedan representar en un plano, el vector de datos que se obtendrá en dicha columna debe tener la misma dimensión que el usado para el eje vertical.
- Etiqueta asociada al eje horizontal: Texto que aparecerá para describir los valores que se representan en el eje vertical.
- Etiqueta asociada al eje vertical: Texto que aparecerá para describir los valores que se representan en el eje vertical.
- Consulta asociada al eje vertical: De la ejecución de esta consulta se obtendrán los datos que conformarán la ordenada vertical de cada punto a representar.
- Columna asociada al eje vertical: Con esta propiedad se especifica la columna a ser usada como vector de ordenadas en el eje vertical. Para que se obtengan pares ordenados que se puedan representar en un plano, el vector de datos que se obtendrá en dicha columna debe tener la misma dimensión que el usado para el eje horizontal.

- 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.

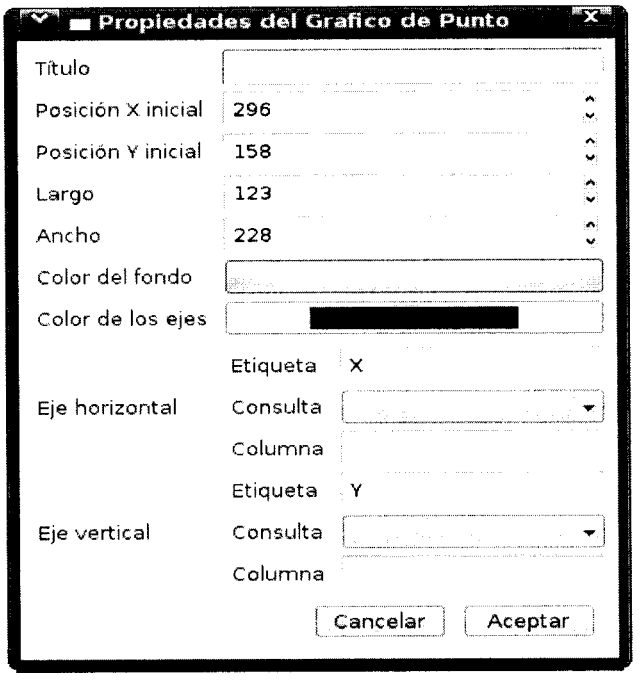


Fig. 7 Interfaz para establecer las propiedades de un gráfico de barras, puntos y líneas

Mientras que para el caso de los gráficos de tipo pastel se adicionan al primer conjunto de propiedades las siguientes:

- Consulta asociada: De la ejecución de esta consulta se obtendrán los datos que se usarán para representar en el gráfico.
- Columna: Con esta propiedad se especifica la columna a ser usada como vector a representar.

2.2 Variables

Como los reportes generalmente se usan para consolidar información, se hace necesaria la incorporación de expresiones matemáticas cuya evaluación sirva para lograr este fin. Para permitir ésto, el sistema de generación de reporte del SCADA incorpora el concepto de variable. Mediante la incorporación de variables en un diseño se podrán aplicar algunos cálculos estadísticos sobre los datos que se obtengan en el momento de generación del reporte.

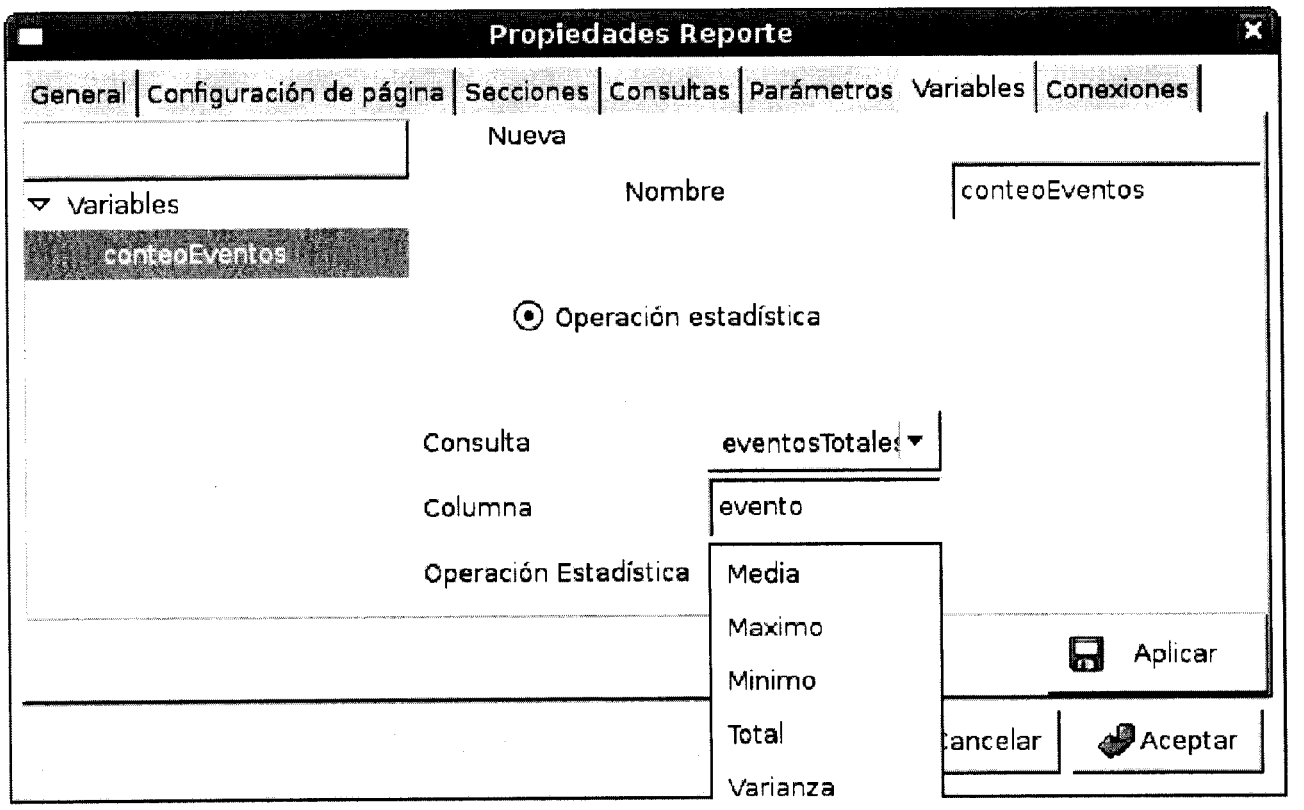


Fig. 8 Edición de variables asociadas a expresiones estadísticas

En esta figura se puede apreciar la lista de variables declaradas en el diseño. Esta interfaz además cuenta con el conformador de expresiones donde se pueden definir los distintos atributos de la variable seleccionada:

- Nombre: Nombre único que identificará la variable en todo el diseño, podrá ser usada para referirse a ella, por ejemplo desde un componte visual de tipo "Campo Calculado".
- Consulta: Consulta que se usará para extraer los datos en una operación de conjuntos de datos, para llenar este campo deben haberse definido con anterioridad las consultas en el editor de consultas.
- Columna: Columna que se usará para una operación de conjuntos y que se obtendrá producto a la ejecución de la consulta SQL.
- Operación estadística: Permite la selección de una operación estadística a aplicar sobre el conjunto de datos que representa el campo "Columna"

Una de las razones fundamentales de realizar operaciones matemáticas sobre los datos es poder visualizar el resultado en el reporte. Para mostrar en un reporte el valor resultante de la

evaluación de una variables se debe incorporar en el diseño correspondiente un componente de tipo "Campo calculado" y asociarlo con dicha variable.
Sin embargo, en un diseño se pueden usar variables para efectuar lo que comúnmente se conoce como cálculos intermedios, o sea cálculos que se emplean como medio para llegar a un resultado final, pero que por sí solos carecen de valor. Variables con la característica anterior no suelen aparecer en el reporte generado con una apariencia visual, sin embargo, si se calculan a la hora de generar el reporte y pueden emplearse para concatenar cálculos.

2.3 Diagrama de clases componente campo de datos

A continuación se muestra la relación que tienen las principales clases que intervienen para que el componente campo de datos funcione correctamente. Como se ve a continuación en la Fig.16 para la lógica del componente participan cinco clases. El DBField le corresponde la visualización de los datos provenientes de la base de datos histórica, al mismo se le especifica la columna que se quiere visualizar del RecorSet y se encarga de toda la pintura o lógica de visualización para mostrar los datos, en el método draw() se maneja todo lo relacionado con dicha lógica, para ello se usa la biblioteca cairo y ahí se maneja todo lo que es fuente de la letra, tamaño de la misma, color, entre otras cosas. La clase Query en relación con la HDBQuery proporciona todo lo que sería conexión con la base de datos, la clase Query maneja el identificador de la conexión y luego la clase RecordSet, encargada de hacer toda la lógica para almacenar los datos que vienen como resultado de la ejecución de una consulta.

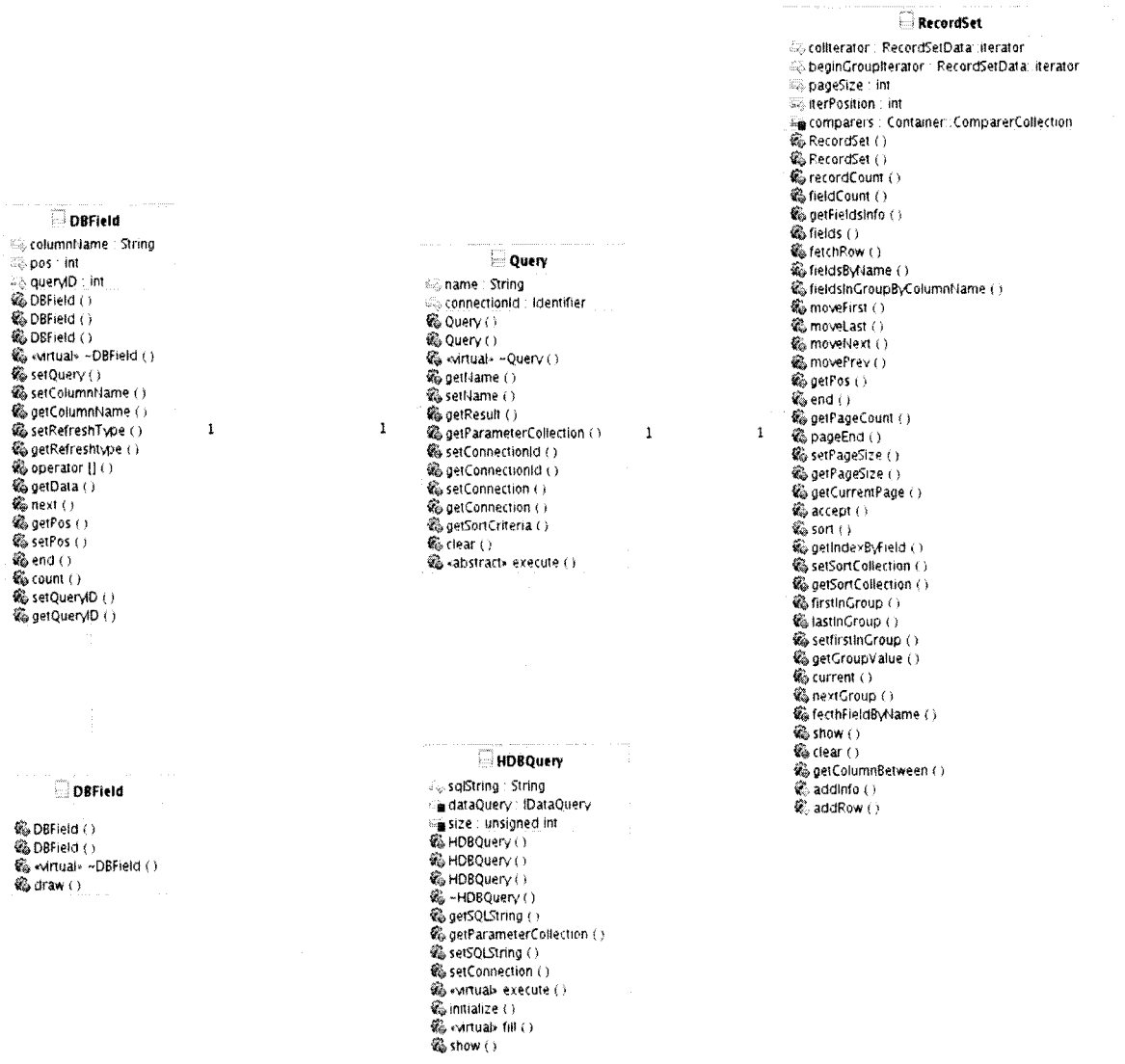


Fig. 9 Diagrama de clases, lógica componente "campo de datos"

2.3.1 Descripción de las principales clases del diagrama diseñado para el componente campo de datos

En las tablas que se muestran a continuación (tabla 1-3), se hace una descripción de las principales clases y sus métodos del componente campo de datos.

Tabla 1 Descripción de la clase Query

Nombre: Query	
Descripción: Clase que provee métodos para realizar consultas a Base de Datos.	
Tipo de clase: Controladora	
Atributo	**Tipo**
name	String
connection	Connection
connectionId	Identifier
recordset	RecordSet
parameters	InputParameterCollection
sortCriteria	SortCollection
Para cada responsabilidad:	
Nombre:	String getName()
Descripción:	Devuelve el nombre de la consulta
Nombre:	void setName(String name)
Descripción:	Asigna nuevo nombre a la consulta
Nombre:	RecordSet* getResult()
Descripción:	Devuelve los datos de la consulta
Nombre:	InputParameterCollection& getParameterCollection ()
Descripción:	Devuelve la colección de parámetros de entrada
Nombre:	void setConnectionId(Identifier connectionId)
Descripción:	Asigna nuevo Identificador de conexión
Nombre:	Identifier getConnectionId ()
Descripción:	Devuelve el identificador de la conexión
Nombre:	void setConnection (Connection* connection)
Descripción:	Asigna nueva conexión
Nombre:	Connection* getConnection ()
Descripción:	Devuelve la conexión
Nombre:	SortCollection& getSortCriteria ()
Descripción:	Devuelve la colección de criterios de ordenamiento
Nombre:	void clear()
Descripción:	Limpia los datos de la consulta
Nombre:	virtual void execute() = 0

Tabla 2 Descripción de la clase RecordSet

Nombre: RecordSet	
Descripción: Clase que almacena, organiza y manipula los datos que devuelve una consulta.	
Tipo de clase: Controladora	
Atributo	**Tipo**
collection	FieldInfoCollection
iterator	RecordSetData::iterator
iterator	RecordSetData::iterator
pageSize	int
iterPosition	int
Para cada responsabilidad:	
Nombre:	int recordCount()
Descripción:	Devuelve el numero de filas
Nombre:	int fieldCount()
Descripción:	Devuelve el numero de campos o columnas
Nombre:	FieldInfoCollection getFieldsInfo()
Descripción:	Devuelve el nombre de los campos o columnas
Nombre:	FieldCollection fields()
Descripción:	Devuelve todos los campos
Nombre:	FieldCollection fetchRow()
Descripción:	Devuelve todos los campos
Nombre:	FieldCollection fieldsByName(String columnName)
Descripción:	Devuelve el campo por nombre deseado
Nombre:	FieldCollection fieldsInGroupByColumnName(column, groupBegin)
Descripción:	Devuelve el campo por nombre deseado en un grupo
Nombre:	void moveFirst()
Descripción:	Mueve el puntero a la primera fila
Nombre:	void moveLast()
Descripción:	Mueve el puntero a la ultima fila
Nombre:	void moveNext()
Descripción:	Mueve el puntero a la siguiente fila
Nombre:	void movePrev()

Descripción:	Mueve el puntero a la anterior fila
Nombre:	int getPos()
Descripción:	Posición actual del puntero
Nombre:	bool end()
Descripción:	Determina si el puntero llego al final
Nombre:	int getPageCount()
Descripción:	Cantidad de páginas a generar
Nombre:	bool pageEnd ()
Descripción:	Determina si el puntero está al final de una página
Nombre:	void sort ()
Descripción:	Ordena según un criterio las filas
Nombre:	int getIndexByField (const String& fieldName)
Descripción:	Posicion de un campo determinado
Nombre:	void setSortCollection (SortCollection sorts)
Descripción:	Asigna nueva colección de criterios de ordenamiento
Nombre:	SortCollection getSortCollection ()
Descripción:	Devuelve la colección de criterios de ordenamiento
Nombre:	void clear()
Descripción:	Borra todos los datos almacenados

Tabla 3 Descripción de la clase DBField

Nombre: DBField	
Descripción: Clase que define la visualización de los campos que vienen de la Base de Datos.	
Tipo de clase: Controladora	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	void draw(SCADA::Report::Draw::Context cr , int yinit)
Descripción:	Método encargado de darle comportamiento de pintura al DBField.

2.3.2 Fragmentos de código

A continuación se muestra lo que sería la lógica de uno de los métodos de la clase RecordSet, donde dicho método se encarga de devolver todos los datos de la columna del RecordSet

previamente pasada por parámetro, este parámetro es una cadena que se corresponde con el nombre de una de las columnas de la matriz de datos devuelta como resultado de la ejecución de una consulta, donde a dicho método se le pasa como parámetro una cadena y el mismo se encarga de devolver la columna del RecordSet proveniente de la base dato histórico y que se corresponde con el nombre pasado como parámetro, en caso de que no exista la columna pasada se lanza una excepción mostrando que no existe la columna especificada.

```
FieldCollection  RecordSet::fieldsByName(std::string columnName)

    FieldCollection result;
    bool found = false;
    for(int i= 0 ; i < info.size() ; i++ )
    {
            if( info[i] == columnName )
            {
                    found = true;
                    moveFirst();
                    while (collterator != data.end())
                    {
                    result.add((*collterator)[i]);
                    moveNext();
                    }
                    moveFirst();
                    break;
            }
    }
    if( !found )
    {
      std::string error = "El campo "+columnName+" no fue encontrado en la ejecución de esta consulta
      throw new Exceptions::RecordSetException( error );
    }
    return result;
```

En este método se usa la lógica del fieldsByName y su función es devolver el valor correspondiente a una posición en la colección de datos FieldCollection de la columna asignada al DBField para su posterior visualización.

```
String DBField::operator [](int index)
{
        if (query->getResult()!= 0)
        {
                RecordSet* temp = query->getResult();
                if(index >= 0 && index < temp->recordCount())
                {
                        Field field = temp->fieldsByName(columnName)[index];
                        text = field.asString();
                }

        }
        return text;
}
```

2.4 Diagrama de clases componente campo calculado

A continuación se muestra la relación que tienen las principales clases que intervienen para que el componente campo calculado funcione correctamente. En el mismo las principales clases son la clase Variable y la clase Statistic, esta última la encargada de hacer todos los cálculos estadísticos para que luego el componente campo calculado los muestre, es importante mencionar que en caso de querer agregarle más cálculos estadísticos al componente, sería la clase Statistic la encargada de dicha responsabilidad, haciendo uso de la biblioteca matemática GSL antes descrita.

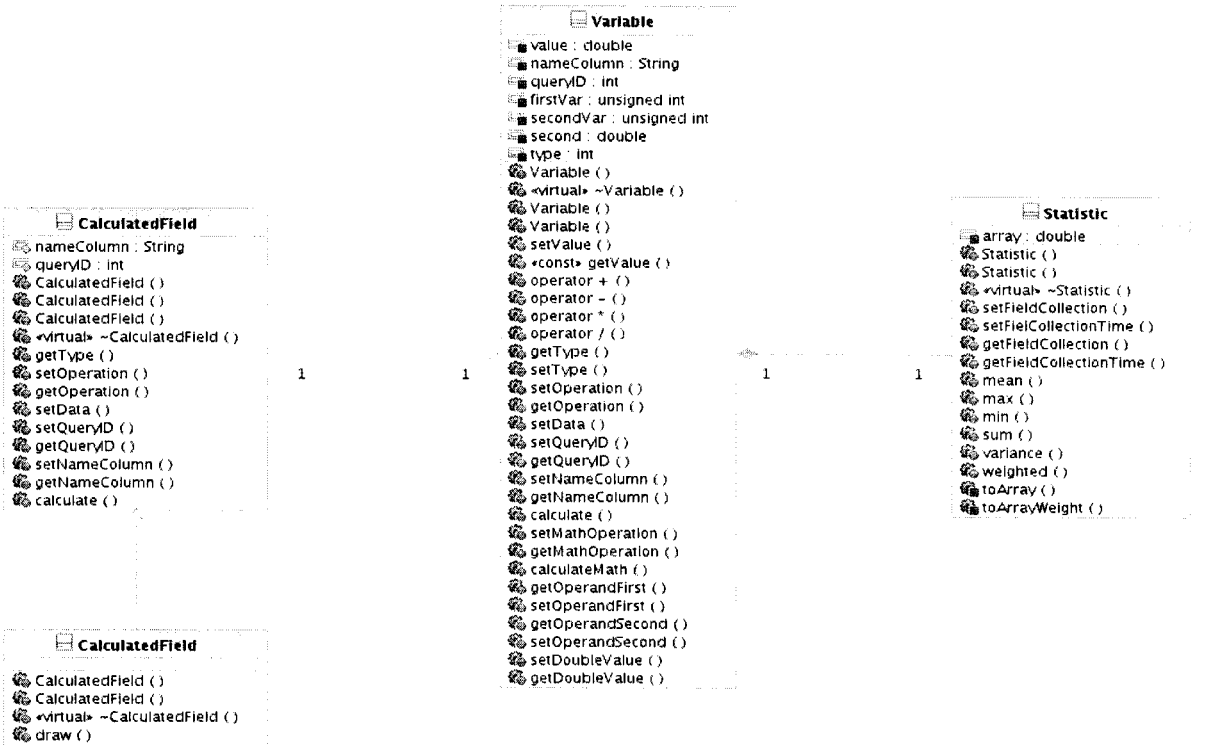


Fig. 10 Diagrama de clases, lógica componente "campo calculado"

2.4.1 Descripción de las principales clases del diagrama diseñado para el componente campo calculado

En las tablas que se muestran a continuación (tabla 4 y 5), se hace una descripción de las principales clases y sus métodos del componente campo calculado.

Tabla 4 Descripción de la clase Variable

Nombre: Variable	
Descripción: Clase responsable de calcular y almacenar el resultado de operaciones matemáticas (suma, resta, multiplicación y división) y operaciones estadísticas (varianza, media ponderada, máximo, mínimos, entre otras).	
Tipo de clase:	
Atributo	**Tipo**
value	double
nameColumn	String
queryID	Int

firstVar	usingned int
secondVar	usingned int
second	double
type	int
Para cada responsabilidad:	
Nombre:	void setValue(double)
Descripción:	Se le asigna al atributo value el valor
Nombre:	void setType (int)
Descripción:	Se le asigna al atributo type el modo de operación que efectuará, tipo 0 : ninguno, tipo 1: estadística y tipo 2 : matemática.
Nombre:	void setOperation ()
Descripción:	Se asigna el tipo de operación estadística.
Nombre:	void setData (fieldCollection)
Descripción:	Se le asigna la colección de datos para el cálculo estadístico.
Nombre:	void setQueryId (int)
Descripción:	Se le asigna al atributo queryID el id de la consulta.
Nombre:	void setNameColumn(String column)
Descripción:	Se le asigna un nuevo campo de la consulta relacionada para el cálculo
Nombre:	estadístico.
Descripción:	void calculate()
Nombre:	Se realiza el cálculo estadístico.
Descripción:	void setMathOperation(MathOperation operation)
Nombre:	Se asigna el tipo de operación matemática a realizar.
Descripción:	void setOperandFirst(Identifier variable)
Nombre:	Se asigna el primer operando para el calculo matemático.
Descripción:	void setOperandSecond(Identifier variable)
Nombre:	Se asigna el segundo operando para el cálculo matemático.
Descripción:	void setDoubleValue(double value)
	Se asigna el segundo operando para el cálculo matemático.

Tabla 5 Descripción de la clase Statistic

Nombre: Statistic
Descripción: Clase controladora encargada de hacer los cálculos estadísticos del reporte.
Tipo de clase: Controladora

Atributo	**Tipo**
data	FieldCollection
dataTime	FieldCollection
array	double*
Para cada responsabilidad:	
Nombre:	void setFieldCollection (FieldCollection data)
Descripción:	Se le asigna la colección de campos al atributo data.
Nombre:	void setFielCollectionTime (FieldCollection data)
Descripción:	Se le asigna la colección de campos al atributo dataTime.
Nombre:	FieldCollection getFieldCollection ()
Descripción:	Se retorna la colección de campos del atributo data.
Nombre:	FieldCollection getFieldCollectionTime()
Descripción:	Se retorna la colección de campos del atributo dataTime.
Nombre:	double mean ()
Descripción:	Se calcula y se retorna la media.
Nombre:	double max ()
Descripción:	Se calcula y se retorna el máximo.
Nombre:	double min ()
Descripción:	Se calcula y se retorna el mínimo.
Nombre:	double sum ()
Descripción:	Se calcula y se retorna la suma.
Nombre:	double variance ()
Descripción:	Se calcula y se retorna la varianza.
Nombre:	double weighted ()
Descripción:	Se calcula y se retorna la media ponderada.

2.4.2 Fragmentos de código

En este fragmento de código perteneciente al método calculate de la clase variable se hace una llamada a una de las operaciones estadísticas que encontramos en Statistic:

```
Case VARIANCE:
 value = statistic.variance();
 break;
```

El método calculate tiene como función o responsabilidad hacer cada una de las operaciones estadísticas usadas en los reportes, ya sean el cálculo de máximos, mínimos o la varianza. Dichos cálculos se hacen sobre los datos provenientes de la base de datos. En caso de que se quisiera poner a los reportes más cálculos estadísticos se le agregarían a dicha clase y luego se les hiciera la llamada desde el método calculate(). Es importante mencionar que los diferentes cálculos se hacen a partir de una colección de datos (FieldCollection), se recorre toda la colección y se hace el cálculo correspondiente.

```
void Variable::calculate()
    {
        if(type == 2)
        {
            if ( statistic.getFieldCollection().size() != 0)
            {
                switch (operation)
                {

                    case MAX:
                        value = statistic.max();
                        break;
                    case MIN:
                        value = statistic.min();
                        break;
                    case SUM:
                        value = statistic.sum();
                        break;
                    case COUNT:
                        value = statistic.count();
                        break;
                    case MEAN:
                        value = statistic.mean();
                        break;
                    case VARIANCE:
                        value = statistic.variance();
                        break;
                }
            }
        }
    }
```

A continuación se muestra un fragmento de código de la clase estadística, ahí se ve cada uno de los métodos de cálculo estadístico que se llaman desde la clase Variable y el método calculate() : **void Variable::calculate()**

```
class Statistic
{
 public:
        Statistic();
        Statistic( FieldCollection data );
        virtual ~Statistic();
        void setFieldCollection ( FieldCollection data );
        void setFielCollectionTime ( FieldCollection data );
        FieldCollection getFieldCollection ();
        FieldCollection getFieldCollectionTime();
 double mean ();
        double max ();
        double min ();
        double sum ();
        double variance ();
        double weighted ();
        int count();

  private:

        double* toArray ( FieldCollection data );
 double* toArrayWeight ( FieldCollection data );
        FieldCollection data;
        FieldCollection dataTime;
        double* array;

};
```

En el método que se muestra a continuación se pone como ejemplo cómo se hace el cálculo de la media ponderada, usando la biblioteca GSL. En caso de que haya alguna falla entonces se lanza una excepción.

```
double Statistic::weighted ()
{
 double* datarray = toArrayWeight( dataTime );
 double* datarrayvalue = new double [data.size()];

for (int j = 0; j < data.size(); j++)
    datarrayvalue[j] = data[j].asDouble();
```

```
if( data.size() )
{
    return gsl_stats_wmean( datarray, 1, datarrayvalue, 1, data.size() );
}
else throw new Exceptions::StatisticException ("The data size could't be 0");
}
```

2.5 Diagrama de clases componente gráfico

A continuación se muestra la relación que tienen las principales clases que intervienen para que el componente gráfico funcione correctamente. En el mismo la principal clase es Chart de la cual heredan los distintos tipos de gráficos (línea, pastel, punto y barra). La implementación y lógica de pintura de todos los tipos de gráficos se implementa o en la clase de PieChart para el caso del gráfico de pastel, y en el caso de los demás gráficos en la clase MultiChart. Es importante mencionar que para la construcción de cualquiera de los gráficos lo único que hay que pasarle como parámetros es la colección de datos proveniente de la base datos que se quiera graficar. El método más importante en cada uno de los gráficos es el draw(), puesto que el mismo en los distintos gráficos controla la lógica para que cada uno de ellos, ya sea de pastel, línea, punto o barra el método se encarga de que se pinten de una manera correcta. Todos se pintan usando la biblioteca gráfica de cairo.

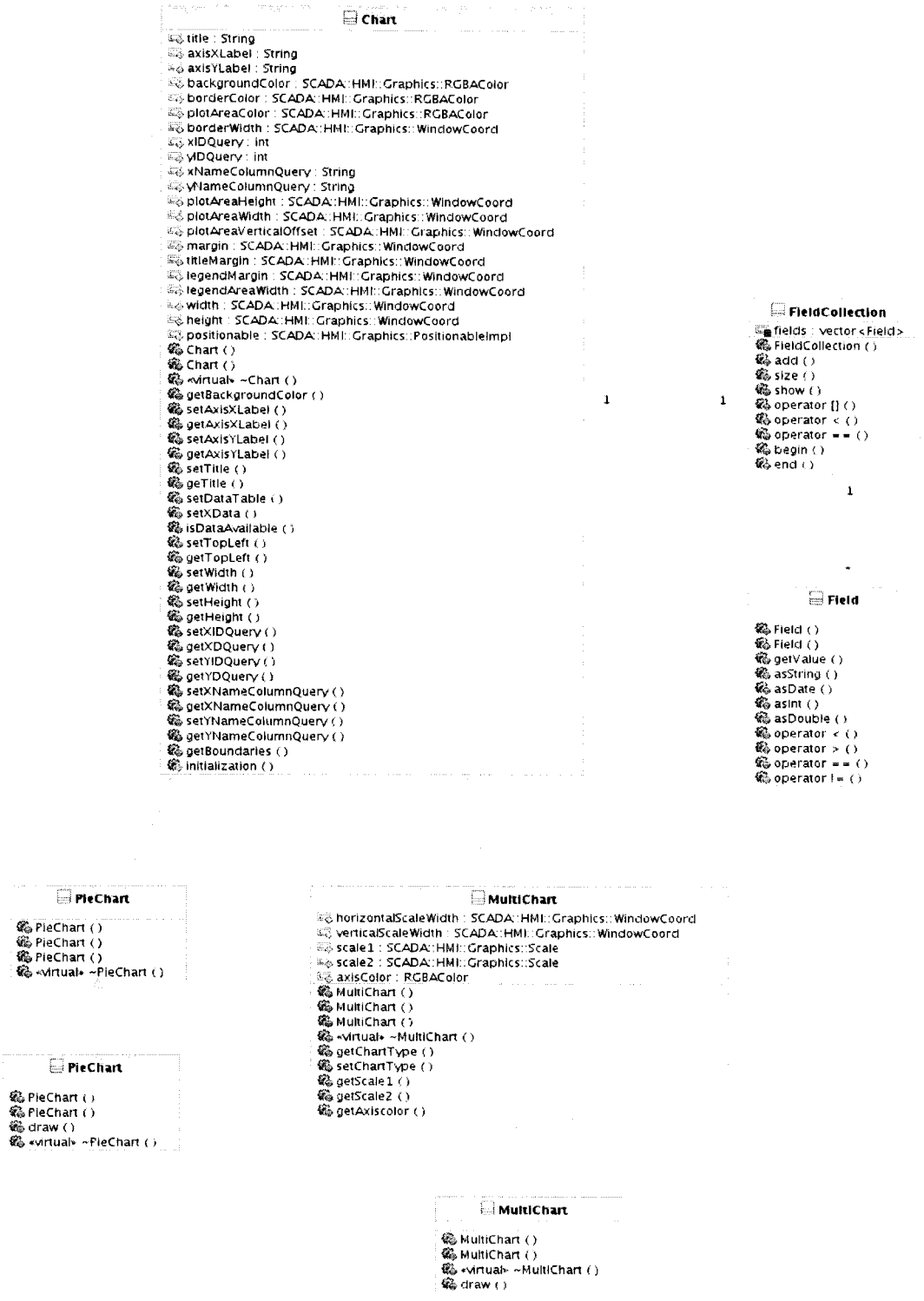


Fig. 11 Diagrama de clases, lógica componente "gráficos"

2.5.1 Descripción de las principales clases del diagrama diseñado para el componente gráfico.

En las tablas que se muestran a continuación (tabla 5 y 6), se hace una descripción de las principales clases y sus métodos del componente campo calculado.

Tabla 6 Descripción de la clase Chart

Nombre: Chart	
Descripción: Clase que representa todos los tipos de gráficos del reporte.	
Tipo de clase: Entidad	
Atributo	**Tipo**
title	String
axisXLabel	String
axisYLabel	String
windowTopLeft	Point
backgroundColor	RGBAColor
borderColor	RGBAColor
plotAreaColor	RGBAColor
borderWidth	WindowCoord
dataTable	FieldCollection
xData	FieldCollection
xIDQuery	unsigned int
yIDQuery	unsigned int
xNameColumnQuery	String
yNameColumnQuery	String
boundingRect	BoundingRect
plotAreaHeight	WindowCoord
plotAreaWidth	WindowCoord
plotAreaVerticalOffset	WindowCoord
margin	WindowCoord
titleMargin	WindowCoord
legendMargin	WindowCoord
legendAreaWidth	WindowCoord
width	WindowCoord

height positionable	WindowCoord PositionableImpl
Para cada responsabilidad:	
Nombre:	RGBAColor& getBackgroundColor ( )
Descripción:	Retorna el valor del atributo backgroundColor.
Nombre:	void setAxisXLabel(String axisXLabel)
Descripción:	Se le asigna un nuevo valor al atributo axisXLabel.
Nombre:	String getAxisXLabel ( )
Descripción:	Se retorna el valor del atributo axisXLabel.
Nombre:	void setAxisYLabel(String axisYLabel)
Descripción:	Se le asigna un nuevo valor al atributo AxisYLabel.
Nombre:	String getAxisYLabel ( )
Descripción:	Se retorna el valor del atributo axisYLabel.
Nombre:	void setTitle(String title)
Descripción:	Se le asigana valor al atributo title.
Nombre:	String geTitle( )
Descripción:	Se retorna el valor del atributo title.
Nombre:	void setDataTable(FieldCollection dataTable)
Descripción:	Se le asigna la colección de campos al dataTable.
Nombre:	void setXData(FieldCollection xData)
Descripción:	Se le asigna la colección de datos al atributo xData.
Nombre:	void setWidth(SCADA::HMI::Graphics::WindowCoord width)
Descripción:	Se le asigna valor al atributo width.
Nombre:	void setHeight(SCADA::HMI::Graphics::WindowCoord height)
Descripción:	Se le asigna valor al atributo height.
Nombre:	void setTopLeft(SCADA::HMI::Graphics::Point* topLeft)
Descripción:	Asigna un nuevo valor al atributo topLeft
Nombre:	Point* getTopLeft()
Descripción:	Retorna un nuevo valor del atributo topLeft
Nombre:	void setXIDQuery(int xIDQuery)
Descripción:	Se le asigna un nuevo valor de id al atributo xIDQuery
Nombre:	int getXDQuery()
Descripción:	Se retorna un nuevo valor del atributo xIDQuery

Nombre:	void setYIDQuery(int yIDQuery)
Descripción:	Se le asigna un nuevo valor al atributo yIDQuery
Nombre:	int getYDQuery()
Descripción:	Se retorna un nuevo valor del atributo yIDQuery
Nombre:	void setXNameColumnQuery(String xNameColumnQuery)
Descripción:	Se le asigna un nuevo valor al atributo xNameColumnQuery
Nombre:	String getXNameColumnQuery()
Descripción:	Se retorna un nuevo valor del atributo xNameColumnQuery
Nombre:	void setYNameColumnQuery(String yNameColumnQuery)
Descripción:	Se le asigna un nuevo valor al atributo yNameColumnQuery
Nombre:	String getYNameColumnQuery()
Descripción:	Se retorna un nuevo valor del atributo yNameColumnQuery

Tabla 7 Descripción de la clase FieldCollection

Nombre: FieldCollection	
Descripción: Almacena los datos de una columna anteriormente especificada del resultado de la ejecución de una consulta.	
Tipo de clase: Entidad	
Atributo	**Tipo**
fields	Vector <Field>
Para cada responsabilidad:	
Nombre:	Field operador [] (int)
Descripción:	Sobrecarga el operador [] para poder saber el valor de una posición
Nombre:	Void add(Field)
Descripción:	Adiciona campos a la colección.
Nombre:	int size()
Descripción:	Devuelve el tamaño de la colección.

2.5.2 Fragmentos de código

A continuación se muestra la lógica de pintura del gráfico de pastel, cada una de las líneas que contiene la palabra cairo como por ejemplo (cairo_translate) es una llamada que se le hace a la

biblioteca gráfica de cairo antes explicada, todo el fragmento de código que se muestra a continuación hace uso de dicha biblioteca.

```
double radius = (std::min(this->plotAreaWidth, this->plotAreaHeight))/2.0f;
cairo_translate( cr, margin, margin + this->plotAreaVerticalOffset);
 cairo_save(cr);

 cairo_translate( cr,(this->plotAreaWidth/2.0f),
                  ((this->plotAreaHeight/2.0f)+10));
 cairo_scale(cr, 1, 0.8);
void PieChart::draw( SCADA::Report::Draw::Context cr , int xOffset, int yOffset)
{
cairo_save(cr);
double sum = 0;
 for( int i = 0; i < dataTable.size(); i++ )
 {
  sum += this->dataTable[i].asInt();
 }

 double init = 0;
 double final = 0;

 for(int k = 1; k < 10; k++)
 {
  final = sum;
  for( int l = dataTable.size() -1; l >= 0; l--)
  {
   double current = this->dataTable[l].asInt();
       init = final - current;

   cairo_save(cr);
       cairo_rotate(cr, init*360/sum*3.141592653589793/180 );
       cairo_move_to(cr, 0, 0 );
       cairo_line_to(cr, radius-55, 0 );
       cairo_arc( cr, 0,
                      0,
            radius-55,
                      0,current*360/sum*3.141592653589793/180);


       cairo_close_path(cr);
       int j = l % 12;
       if(k != 9)
       {
    cairo_set_source_rgb( cr, 0.8 * static_cast<double>(colorTable[j].getRed()/255.0f),
```

```
                                        0.8 * static_cast<double>(colorTable[j].getGreen()/255.0f),
                                        0.8 * static_cast<double>(colorTable[j].getBlue()/255.0f) );
        }
    else
        {
         cairo_set_source_rgb( cr, 1, 1, 1);
         cairo_stroke_preserve( cr );
         cairo_set_source_rgb( cr, static_cast<double>(colorTable[j].getRed()/255.0f),
                                        static_cast<double>(colorTable[j].getGreen()/255.0f),
                                    static_cast<double>(colorTable[j].getBlue()/255.0f) );

    }
         cairo_fill(cr);
         if(k == 9)
         {
          cairo_save(cr);
            cairo_rotate(cr, current*360/sum/2.0*3.141592653589793/180 );
            cairo_move_to(cr, radius-50, 0);
            cairo_rel_line_to(cr, 16, 0);
            cairo_set_source_rgb( cr, 0, 0, 0);
            cairo_stroke(cr);
            cairo_move_to(cr, radius-28, 0);
            cairo_save(cr);
              cairo_identity_matrix(cr);
                  cairo_set_operator(cr, CAIRO_OPERATOR_ATOP);
                  cairo_show_text(cr, (char*) boost::lexical_cast<string>(current).c_str() );
            cairo_restore(cr);

         cairo_restore(cr);

          }
         cairo_restore(cr);
            final = init;
    }
         cairo_translate(cr, 0, -1);
   }
cairo_restore(cr);
 }
```

2.6 Ejemplos de los componentes gráficos y campo de datos

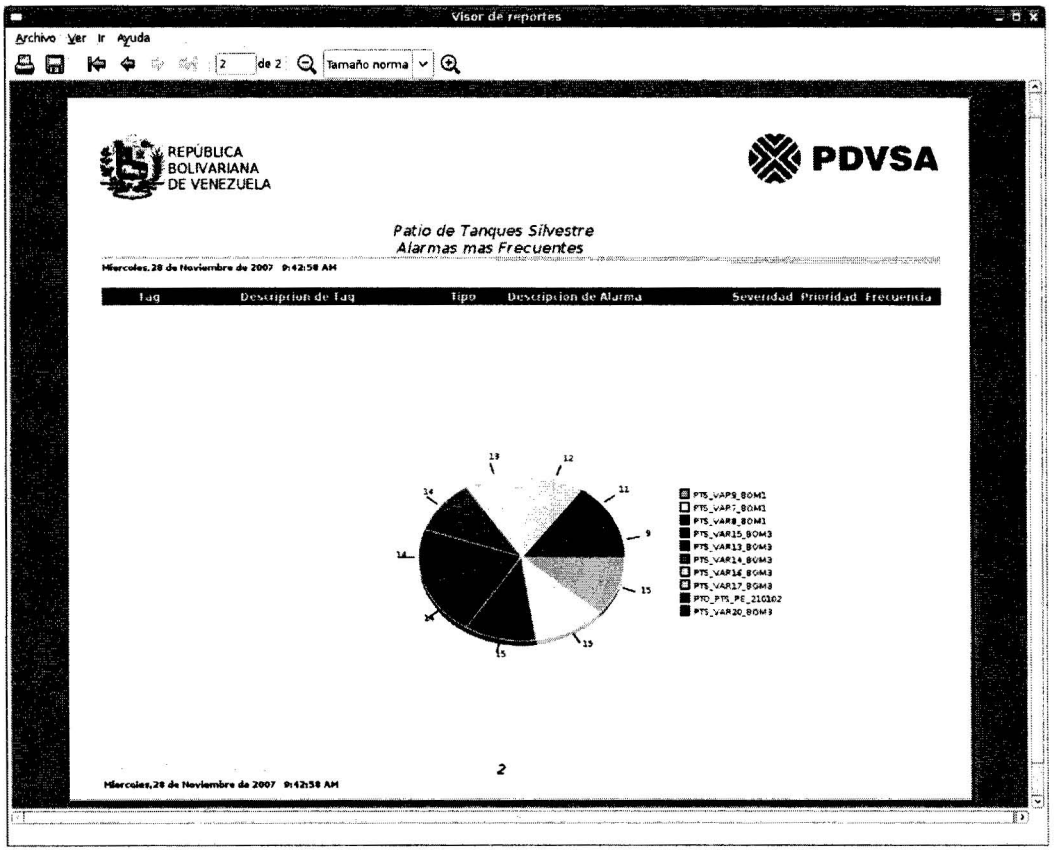


Fig. 12 Reporte generado sobre la interfaz de visualización, componente gráfico

Como se muestra en la figura anterior se puede visualizar lo que sería un reporte con uno de los componentes tratados en este trabajo, en este caso es un gráfico de pastel. A este componente se le asocian todas las propiedades de las cuales se habló anteriormente. Éste es considerado uno de los componentes más importante y complejos con que cuenta el editor de reportes, ya que con el mismo se puede ilustrar de una mejor manera el operario o usuario del sistema el comportamiento de determinada área en la industria.

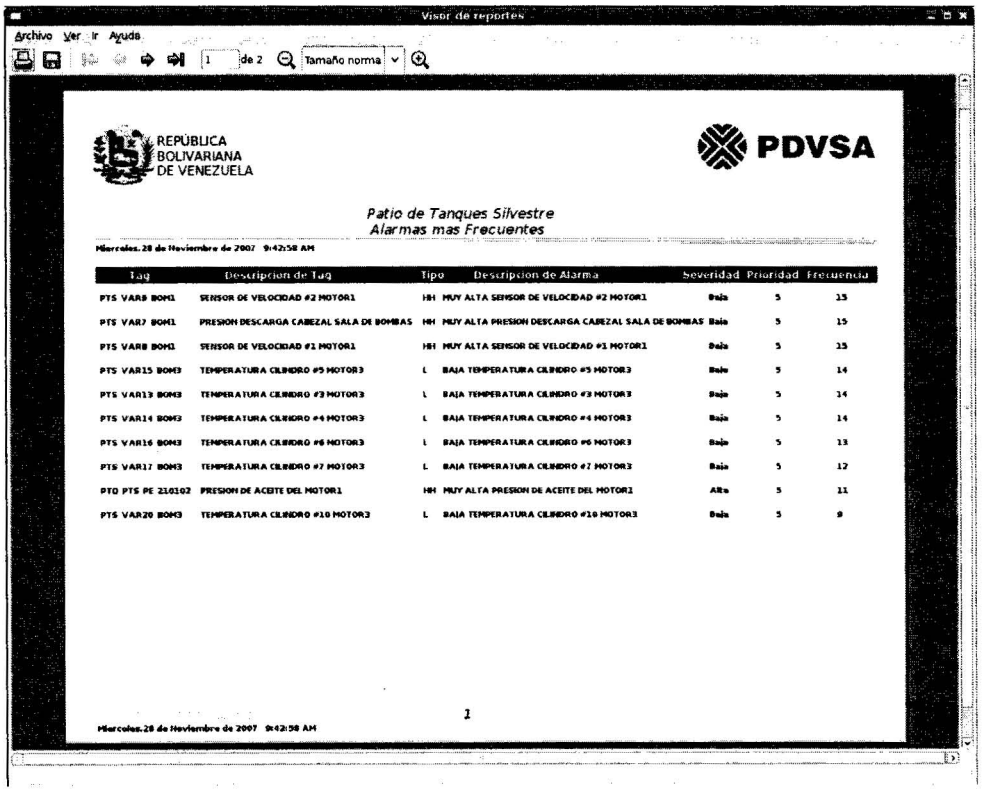


Fig. 13 Reporte generado sobre la interfaz de visualización, componente campo de datos

En la anterior figura (Fig.13), se muestra un reporte generado de las alarmas más frecuentes donde se muestra el comportamiento visual del componente campo de datos, todas las columnas que se visualizan en dicho reporte provienen de una base de datos históricos y usan el componente mencionado para su visualización, campo de datos es considerado lo más importante en la generación de reportes puesto que sin él no se podrían mostrar datos de ninguna de las consultas hechas por ninguno de los operarios o usuarios del sistema.

Capitulo 3: Validación de la solución

Introducción

Una vez elaborados los diferentes componentes de manejo campo calculado, campo de datos y gráficos, se hace necesario llevarlos a pruebas, puesto que el funcionar correctamente no significa que no carezcan de fallas, es por ello que este capítulo se centra en las pruebas. Para de una manera más eficientes validar cada una de las soluciones que se le dieron en la implementación. Es así que a las diferentes clases y métodos empleados para darle solución a los diferentes componentes se les lleva a la realización de pruebas unitarias en el siguiente capítulo.

3.1 ¿Qué es la fase de prueba?

La fase de pruebas es la que añade al producto final el valor para afirmar que ya se ha alcanzado la calidad requerida. Gran porciento de los programas que se desarrollan tienen errores, y es en la fase de pruebas donde se descubren, ese es el valor que añade esta etapa, el objetivo específico de la fase de pruebas es encontrar cuantos más errores mejor. Es por ello que probar es una de las fases más importantes para que un producto salga con la calidad máxima y sin errores.

3.2 Pruebas de Unidad

Las pruebas de unidad es la manera de comprobar el funcionamiento correcto de determinado módulo de código y ayuda a independizar el módulo, significa esto que se pueden probar los módulos independientemente uno de otros.

Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del módulo. Si los datos no fluyen correctamente, todas las demás pruebas no tienen sentido. (PRESSMAN 2005)

Para que una prueba unitaria sea buena se deben cumplir los siguientes requisitos:

- **Automatizable**: no debería requerirse una intervención manual. Esto es especialmente útil para integración continúa.
- **Completas**: deben cubrir la mayor cantidad de código.
- **Repetibles o Reutilizables**: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También es útil para integración continua.
- **Independientes**: la ejecución de una prueba no debe afectar a la ejecución de otra.

- **Profesionales**: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, entre otras cosas.

El objetivo de las pruebas unitarias es aislar, cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el segmento de código debe satisfacer. Estas pruebas aisladas hacen posible cinco ventajas básicas:

1. **Fomentan el cambio**: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
2. **Simplifica la integración**: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.
3. **Documenta el código**: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
4. **Separación de la interfaz y la implementación**: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
5. **Los errores están más acotados y son más fáciles de localizar**: dado que tenemos pruebas unitarias que pueden desenmascararlos.

3.2.1 Recursos empleados para la realización de pruebas.

Para realizar las pruebas se utilizaron como recursos físicos: computadoras con un microprocesador Dual Core con una velocidad del CPU de 1.66 Ghz, la memoria RAM que se dispuso fue de 1 Gb y un disco duro con capacidad de 120 Gbytes.

Se empleó el sistema operativo GNU/Linux, se utilizó la librería de pruebas CxxTest y el IDE de desarrollo fue Eclipse 3.2 con el plug-in CDT 2.1 para C++.

3.3 Pruebas realizadas a las clase Query

Clase TestQuery
Prueba unitarias de la clase: ***Query***
Casos de prueba del método: **void setConnectionId(Identifier connectionId)**
Variables a considerar en el caso de prueba: ***Identifier connectionId***

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

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que el identificador esté entre los rangos de los valores correctos.	Válida
2	Verificar que el identificador lanza un error o excepción.	Inválida

Casos de Prueba:

1-*void testSetConnection_1 ()*

2-*void testSetConnection_2 ()*

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 *connectionId*.	180	180	180	
2	Verificar que se lanza una excepción o mensaje de error cuando el identificador es "cero" o "nulo".	0	Se lanza un mensaje de error o excepción.	No se muestra ningún mensaje de error o excepción.	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de un identificador con "cero" o "nulo".

Recomendación

Mostrar un mensaje de error o excepción para los casos en los que se entren valores no válidos.

Prueba unitarias de la clase: ***Query***

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
2	Verificar que no realice cambio del valor del atributo cuando el dato entrado es una cadena no válida.	Inválida
3	Verificar que no realice cambio del valor del atributo cuando el dato entrado no es una cadena	Inválida

Nota: Cadena no válida es la que contiene caracteres especiales (Ej. &, ·, @, #, etc.)

Casos de Prueba:

1-*void testSetName()*

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Se pone a prueba que se pueda cambiar el valor del atributo	Un valor de cadena	El valor de la cadena pasada como parámetro	El valor de la cadena pasada como parámetro	

2	Se pone a prueba intentar cambiar el valor del atributo entrando una cadena con caracteres especiales	Un valor de cadena con caracteres especiales	Excepción o algún tipo de notificación por parte de objeto cuando se le pasa como argumento una cadena con caracteres espaciales	No se produce ninguna notificación ni excepción.	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de una cadena con caracteres especiales
3	Se pone a prueba intentar cambiar el valor del atributo entrando un valor no cadena (Ej. Un float)	Un valor no cadena	Excepción o algún tipo de notificación por parte de objeto cuando se le pasa como argumento un valor diferente a una cadena	No se produce ninguna notificación ni excepción.	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de un valor diferente a una cadena

Nota: Valor no cadena es otro tipo de datos que pudiera romper la lógica de un nombre, ejemplo no cumple objetivo poner nombres con números, por tanto estaría incorrecto pasar tipos de datos (int, float, otros).

Recomendación

Realizar validaciones para valores que no sean cadenas o para las cadenas con caracteres especiales.

Prueba unitarias de la clase: ***Query***

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

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

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

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Se le asignará al atributo *connectionId* un valor pasado como parámetro	Válida

Casos de Prueba:

1-***void*** **testgetConnectionId ()**

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que el valor devuelto por el método sea el mismo pasado como parámetro	120	120	120	

3.4 Pruebas Realizadas a la clase Chart

Clase TestChart

Prueba unitarias de la clase: ***Chart***

Casos de prueba del método: **void setDataTable(FieldCollection dataTable)**

Variables a considerar en el caso de prueba: ***FieldCollection dataTable***

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

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Se le asignará al atributo *dataTable* una colección de datos (FieldCollection) pasada como parámetro	Válida

Casos de Prueba:

1- **void testsetDataTable(FieldCollection dataTable)**

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Verificar que la colección de datos pasada como parámetro es la correcta.	2,108,5,10	2,108,5,10	2,108,5,10	

Casos de prueba del método: ***void setXIDQuery(unsigned int xIDQuery)***

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

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

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
2	El valor de la variable estará fuera los valores permisibles de la variable.	Inválida

Casos de Prueba:

1-***void testSetXIDQuery_1 ()***

2-***void testSetXIDQuery_2 ()***

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Que se le asigne correctamente el valor del color pasado como parámetro.	25	25	25	

2	Que se le asigne a la variable un valor fuera del rango	-10	0	187	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de valores negativos

Recomendación

Realizar validaciones para valores negativos, además de mostrar mensajes de error o excepción para estos casos.

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	Se pasará un valor que sea correcto para que el gráfico adopte un valor de ancho.	Válida

Casos de Prueba:

1-***void testSetWidth ()***

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Verificar que se devuelvan los datos esperados a partir de un valor inicial entrado como	75	75	75	

	parámetro.				

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

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

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

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

Casos de Prueba:

1-***void test* getBackgroundColor *_1 ()***

2-***void test* getBackgroundColor *_2 ()***

Caso de Prueba	**Objetivo de la Prueba**	**Datos de Entrada**	**Salida Esperada**	**Salida Obtenida**	**Observación**
1	Que se le asigna correctamente el color	RGBA color (255,255,255,0)	Color con los mismos valores pasados en	Color con los mismos valores pasados en	

	pasado como parámetro.		los datos de entrada.	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.	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de valores negativos

Recomendación

Realizar validaciones para valores negativos, además de mostrar mensajes de error o excepción para estos casos.

3.5 Pruebas Realizadas a la clase Variable

Prueba unitarias de la clase: ***Variable***

Casos de prueba del método: **void setValue(double value)**

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

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

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

Casos de Prueba:

1-*void test*setValue *()*

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Se pone a prueba que se pueda cambiar el valor del atributo	Un valor de cadena de tipo double	El valor pasado como parámetro	El valor pasado como parámetro	

Recomendación

Realizar validaciones para valores que no sean cadenas o para las cadenas con caracteres especiales.

Casos de prueba del método: **void setNameColumn(String column)**

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

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

Clase de Equivalencia	Clase de Equivalencia	Clasificación de las Clases de Equivalencia
1	Verificar que cambie el valor del atributo *nameColumn*.	Válida
2	Verificar que no realice cambio del valor del atributo cuando el dato entrado es una cadena no válida.	Inválida

Nota: Cadena no válida es la que contiene caracteres especiales (Ej. &, ·, @, #, etc.)

Casos de Prueba:

1-*void testsetNameColumn*(String column)

Caso de Prueba	Objetivo de la Prueba	Datos de Entrada	Salida Esperada	Salida Obtenida	Observación
1	Se pone a prueba que se pueda cambiar	Un valor de cadena	El valor de la cadena pasada como	El valor de la cadena pasada como	

	el valor del atributo		parámetro	parámetro	
2	Se pone a prueba intentar cambiar el valor del atributo entrando una cadena con caracteres especiales	Un valor de cadena con caracteres especiales	Excepción o algún tipo de notificación por parte de objeto cuando se le pasa como argumento una cadena con caracteres espaciales	No se produce ninguna notificación ni excepción.	No se obtuvo la salida esperada ya que la clase no tiene validada la entrada de una cadena con caracteres especiales

Casos de prueba del método: **void setQueryId (int queryID)**

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

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

Clase de Equivalencia	**Clase de Equivalencia**	**Clasificación de las Clases de Equivalencia**
1	Verificar que el identificador esté entre los rangos de los valores correctos.	Válida
2	Verificar que el identificador lanza un error o excepción.	Inválida

Casos de Prueba:

1-*void testSetQueryId _1 ()*

2-*void testSetQueryId _2 ()*

Caso de	**Objetivo de la Prueba**	**Datos de**	**Salida Esperada**	**Salida Obtenida**	**Observación**

Prueba		**Entrada**			
1	Verificar que se pueda cambiar el valor del atributo *queryID*.	40	40	40	
2	Verificar que se lanza una excepción o mensaje de error cuando el identificador es "cero" o "nulo".	0	Se lanza un mensaje de error o excepción.	No se muestra ningún mensaje de error o excepción.	Se obtuvo la salida esperada ya que la clase tiene validada para un identificador con "cero" o "nulo".

Conclusiones

Con la realización del presente trabajo se arribó a las siguientes conclusiones:

- Se modelaron los componentes campo de datos, campo calulado y gráficos, garantizándose de esta manera una solución más robusta.
- Se implementó cada uno de los componentes a partir del diseño de los mismos logrando así el cumplimiento del objetivo principal del trabajo.
- Se validó la implementación de los componentes mediante la realización de las pruebas unitarias, demostrando de esta manera la robustez de la solución.
- Con el desarrollo de los componentes se logró hacer más interactivo y funcional la herramienta para la edición de plantillas del SCADA.

Recomendaciones

Con la realización del presente trabajo se recomienda:

- Incorporar la funcionalidad de tabulación al componente campo de datos para que de esta manera el usuario tenga mayor interacción con el mismo.
- Incorporar la funcionalidad de gráfica comparativa al componente gráfico para lograr así que el usuario del sistema pueda generar comparaciones entre diferentes colecciones de datos.
- Incorporar nuevas funciones matemáticas y estadísticas al componente campo calculado.
- Crear un componente que dé la funcionalidad para la introducción de párrafos en el ambiente de edición del diseñador de plantillas.

Bibliografía

- The GNU Project, [En Linea]. GNU Project, 2003. [2007]. Disponible en: http://www.gnu.org.
- Aquilino, Rodríguez Penin. *Sistemas SCADA.* 2006.
- Fundación del Software Libre, [En Línea]. FSF, 1999. [2007]. Disponible en: http://www.fsf.org/
- Boost C++ Libraries. [En línea] www.boost.org.
- ALBERTO MOLPECERES TOURIS, M. P. M. Arquitectura empresarial y software libre, J2EE, 18/08/2002. Disponible en: http://www.javahispano.org/articles.article.action?id=70 (05/01/2007)
- ARINA OYAGA, G. informáticos en tiempo real. Ejemplos prácticos. España, Marzo de 2000. p.
- CRYSTALREPORTS.ORG. Crystal Reports XI from Business Objects. Disponible en: http://www.businessobjects.com/products/reporting/crystalreports/start_xi.asp (11/01/2007)
- DISTEFANO., M. Comunicaciones en entornos industriales. . Mendoza. Argentina, Universidad Nacional de Cuyo., 1999. p.
- BANKHACKER.COM. Web Services. Disponible en: http://web-services.bankhacker.com/ (05/01/2007).
- AGATA.ORG. Agata Report. Disponible en: http://www.agata.org.br/us/index.php (11/01/2007)

Referencias bibliográficas

(1) SCADA, N. Ingenieria Conceptual Proyecto Desarrollo SCADA Nacional, 2005.

(2) Plataforma SCADA PDVSA. Venezuela, PDVSA, 2005.

(3) SCADA, S. Red Nacional de Gasoductos; Sistema SCADA. Colombia, Bucaramanga, 2002.

(4) FOUNDATION, O. OPC Historical Data Access Specification. . 1.20., V., 2003.

(5) IEEE. Tutorial Course. Course Text. Fundamental of supervisory control systems; Engineering Societies Library, 18 de febrero 1982.

(6) JAMES JACOBSON, I. B., GRADY Y RUMBAUGH. El Proceso Unificado de Desarrollo de Software.

(7) JASPERREPORTS. Home. Disponible en: http://jasperreports.sourceforge.net/ (05/02/2008)

(8) MICROSOFT. Microsoft Office Online: Access. Disponible en: http://office.microsoft.com/es-es/FX010857913082.aspx (10/02/2008)

(9) MANAGER, R. Report Manager Official page. Disponible en: http://reportman.sourceforge.net/ (07/01/2008)

(10) JFREE.ORG. JFreeReport. Disponible en: http://www.jfree.org/jfreereport/ (07/01/2007)

(11) KOFFICE.ORG. The KOffice Project. Disponible en: http://www.koffice.org/kugar/

(12) LARMAN, C. UML y patrones. Introducción al análisis y diseño orientado a objetos. 2. 2004. p.

(13) MOLPECERES, A. Procesos de desarrollo: RUP, XP y FDD. Disponible en: http://www.javahispano.org/articles.article.action?id=76 (04/02/2008).

(14) OPENOFFICE.ORG. Home. Disponible en: http://www.openoffice.org/ (12/02/2008)

(15) OPENREPORT. Open Source and profesional reporting solution. Disponible en: http://openreport.org/ (07/06/2008)

(16) REKALLREVEALED.ORG. Rekall: The database front-end for KDE and the Web. Disponible en: http://www.rekallrevealed.org/kbExec.py

(17) SCADA, E. P. Especificación de requisitos de software.: Versión 3.0. Venezuela, 2006.

(18) PRESSMAN, R. S. Capítulo 17: Técnicas de Prueba del Software. en: Ingeniería del Software. Quinta Edición. La Habana, Félix Varela, 2005. Parte 1: 601.p.

Anexos

Anexo 1

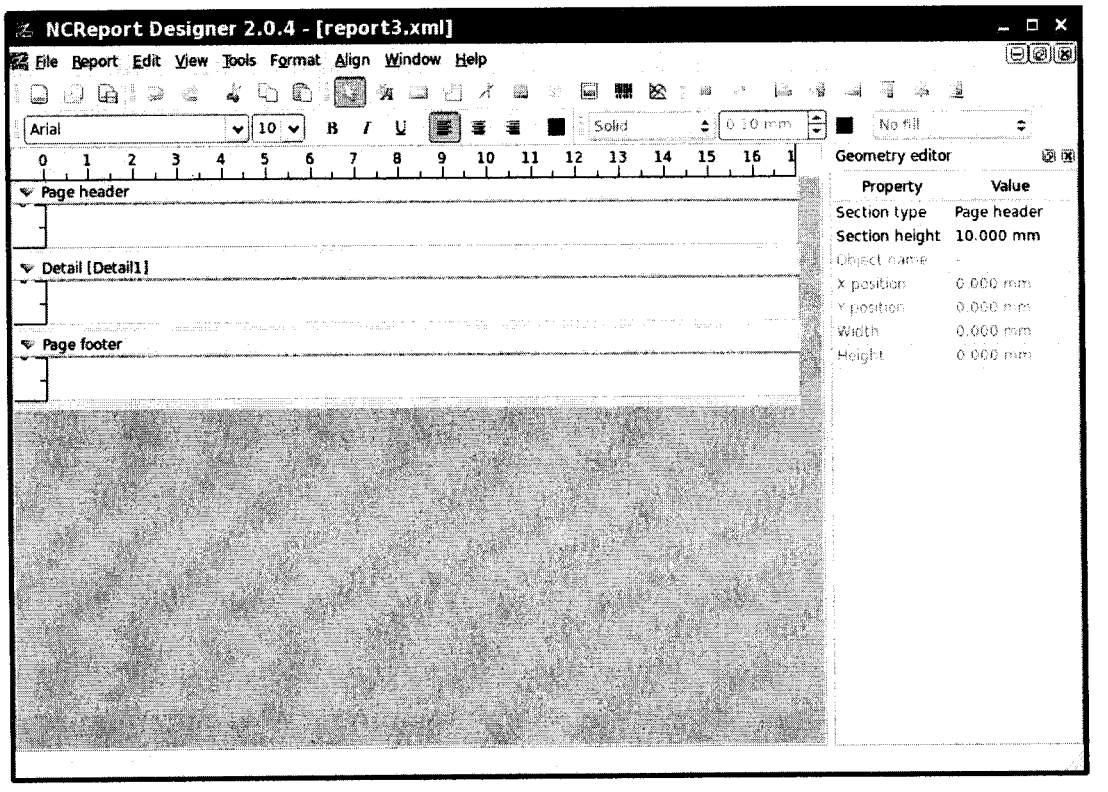


Fig. 14 Ambiente de trabajo en el diseñador de NCReport

Anexo 2

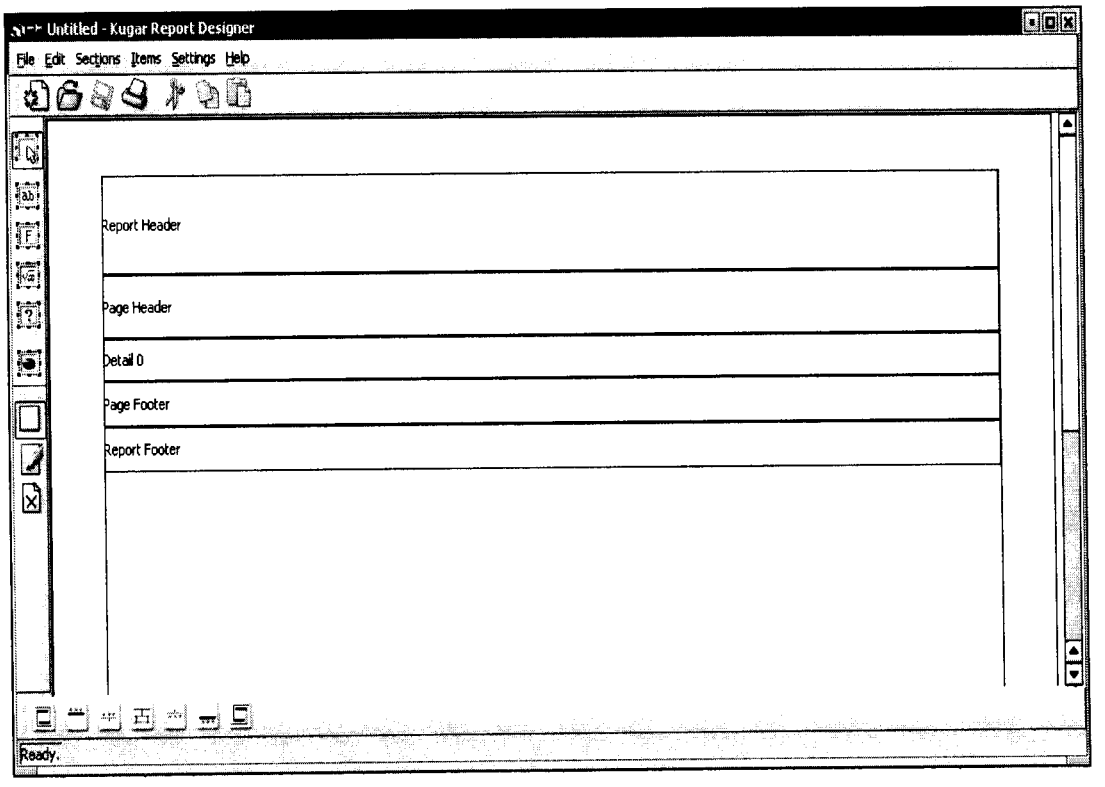


Fig. 15 Vista diseño de Kugar

Anexo 3

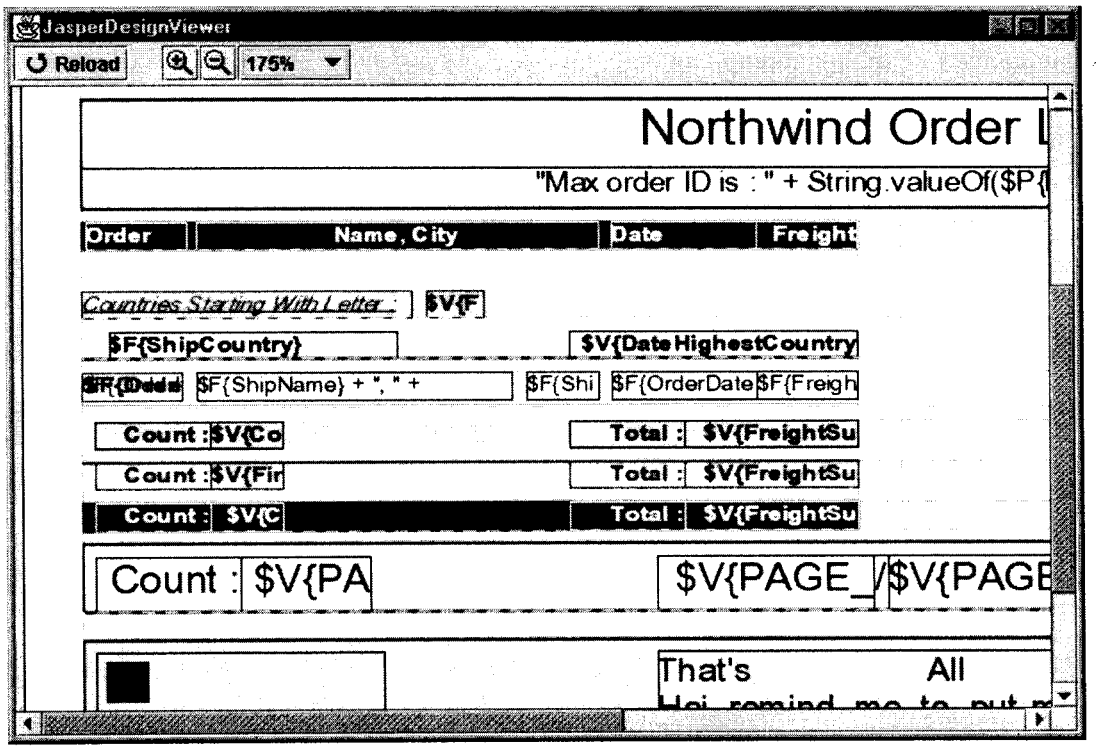


Fig. 16 Ventana de diseño de informe

Anexo 4

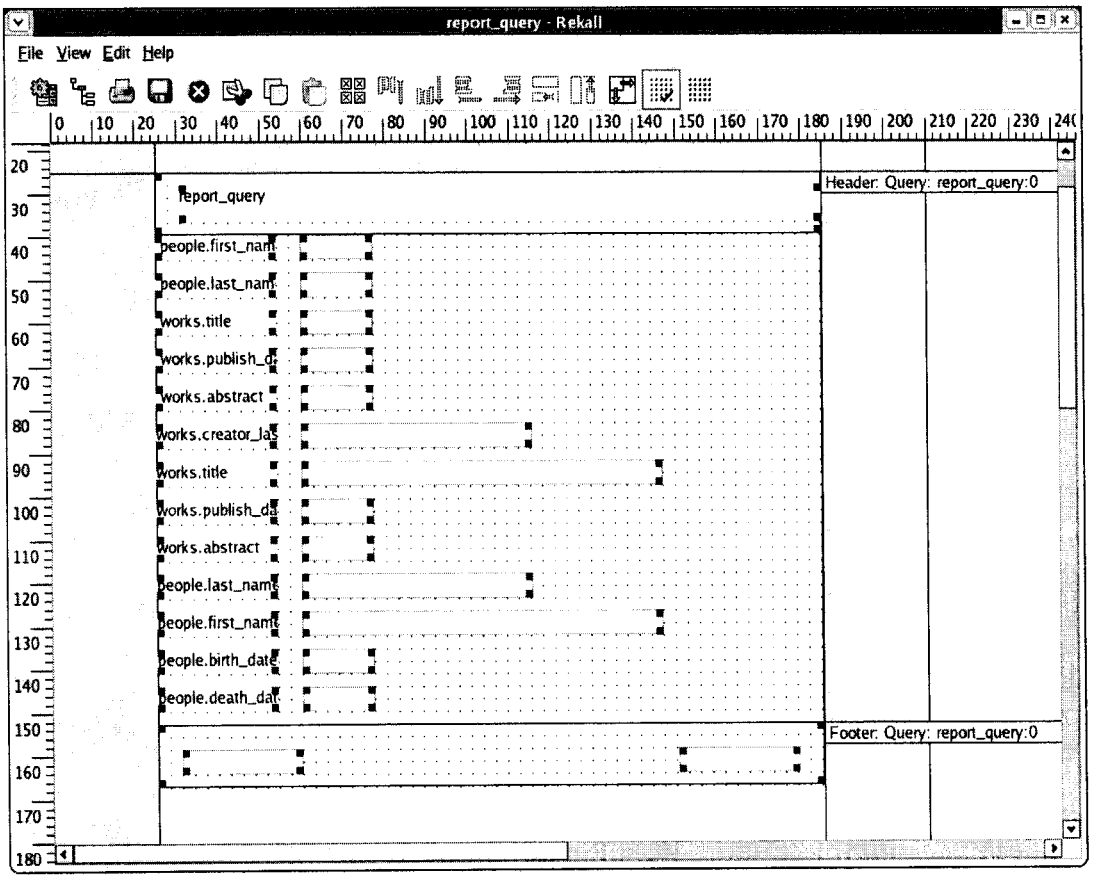


Fig. 17 Ambiente de diseño Rekall

Anexo 5

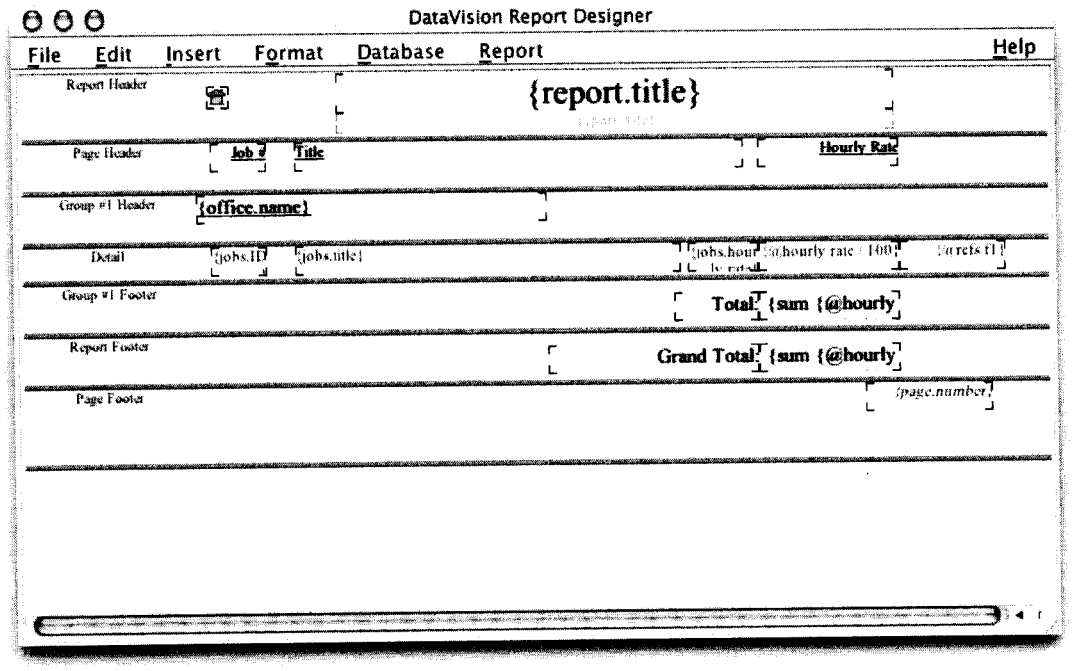


Fig. 18 Ventana de diseño de informe

Glosario de términos

Componente: Objetos gráficos que se encuentran en la paleta del diseñador de reportes utilizados para los diseños de plantillas.

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

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

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

Secciones: 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

Variable: Permite el cálculos estadísticos sobre los datos que se obtengan en el momento de generación del reporte.