Universidad de las Ciencias Informáticas.

Facultad 5

TÍTULO: Análisis de la arquitectura del módulo Interfaz Hombre Máquina del SCADA Nacional.

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

Autor: Jordenys Pérez Feria

Tutor: Ing. René López Baracaldo

Ciudad de la Habana, Julio 2008.
"Año 50 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 5 de la Universidad de las Ciencias Informáticas así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo.

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

_____ _____

Firma del Autor Firma del Tutor

Datos del Contacto

Tutor:

Ing. René López Baracaldo

Email: rene@uci.cu

Años Graduado: 4

Años de Experiencia en el tema: 4

"Los problemas significativos que enfrentamos no pueden ser resueltos con el mismo nivel de pensamiento que teníamos cuando los creamos"

Albert Einstein

Agradecimientos

A Fidel y a la UCI, por hacerme posible formar parte de este proyecto futuro.

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

A Jandrich, por su amistad y ayuda.

A mis compañeros y amigos que juntos hemos transitado por este camino y que de una forma u otra, me han ayudado en la realización de este trabajo.

A la facultad 5 y a mis profesores, por el apoyo que siempre me han brindado y con el cual he logrado terminar mi carrera profesional, siendo para mí, la mejor de las herencias.

A mis abuelos, por ser tan especiales para mí.

A mis padres, por tanto amor y entrega, por inspirarme con tantos ejemplos de genialidad y sacrificio, por estar en el lugar preciso para un consejo oportuno, por el apoyo recibido durante mi formación profesional y a lo largo de toda mi vida.

A mi hermano "Osmy", tú escribirás este trabajo muy pronto. Tengo toda la confianza en ti.

A mi "Lisi", por su ayuda, por su inmenso apoyo, amor y confianza. Por ser la pareja con la que siempre soñé.

Dedicatoria

A mis queridos padres por ser mis guías,

A mi hermano por quererlo tanto,

A mi lisi por estar a mi lado durante estos maravillosos años…

Resumen

En el presente está dirigido al mejoramiento de la arquitectura del módulo Interfaz Hombre-Máquina de un sistema de Adquisición de Datos y Control de Supervisión (SCADA), con requerimientos críticos de tiempo real y fiabilidad. Actualmente existe la necesidad de mejorar la arquitectura implementada en el módulo para minimizar el efecto de una serie de problemas que se están presentando.

Este trabajo tiene como objetivo concreto la modelación de una serie de mejoras para partes específicas del módulo HMI. Para obtener la solución se comienza con una descripción general de los sistemas SCADA. Se hace referencia al surgimiento de la Interfaz Hombre-Máquina (HMI). Se analiza la metodología de desarrollo empleada y se hace una breve descripción de las tecnologías y herramientas de desarrollo utilizadas para el desarrollo del trabajo.

Seguidamente se hace un análisis de la arquitectura que presenta la implementación actual del módulo HMI, las técnicas utilizadas para el diseño y un análisis crítico de ésta a través de las distintas vistas utilizadas para representar la arquitectura. Luego se describe la solución propuesta a los problemas mediante nuevos diseños a secciones específicas de la arquitectura. La solución se expone a través de diagramas de clases y la explicación descripción de éstos. Finalmente se exponen las conclusiones del trabajo y se proponen recomendaciones para la implementación de las propuestas de diseño.

Palabras Claves

Arquitectura, HMI, Diseño, Metodología de Desarrollo, SCADA, Tiempo Real.

Tabla de Contenido

Índice de Figuras.

Introducción

La necesidad de supervisar y controlar a distancia un proceso ha sido una premisa para el hombre moderno. Los primeros mecanismos de supervisión eran sencillos sistemas de telemetría que solo proporcionaban reportes de las condiciones de campo y brindaban parámetros de las unidades remotas.

La llegada de nuevos avances tecnológicos proporcionó cada vez mejores soluciones a los distintos problemas. Las computadoras con mayor capacidad de cálculo son capaces de realizar tareas más complejas permitiendo adicionar gran cantidad de funcionalidades que hacen los sistemas más completos.

Un ejemplo de soluciones cada vez más complejas son los Sistemas de Control Industrial (sus siglas en inglés ICS), término general que encapsula a varios tipos de sistemas de control SCADA.

Hoy día los SCADA son parte integral de la estructura de las industrias y no son vistos como una simple herramienta operacional, sino como un recurso importante de información. Funcionan como un centro de responsabilidad, al punto de poder controlar situaciones excepcionales como desastres ecológicos en caso de industrias que trabajen con materiales contaminantes; pérdidas de vidas humanas en explosiones o escape de gases tóxicos.

Los SCADA proporcionan gran cantidad de datos que se entregan a usuarios fuera del ambiente de control con los que se realizan estudios de comportamientos, detección de irregularidades y toma de decisiones. Las aplicaciones SCADA que se comercializan en el mercado son sistemas propietarios con un alto costo para las empresas que los adquieren, a los clientes no se les proporcionan el código fuente de forma que no pueden adicionar funcionalidades ni modificar las existentes, además de todo esto la seguridad del sistema queda en manos del proveedor del software ya que los usuarios del mismo no tienen conocimiento del funcionamiento interno del programa.

Por los motivos expuestos se ha decidido implementar una aplicación SCADA basado en software libre y en estándares abiertos. En la actualidad, un equipo multidisciplinario de diferentes empresas y universidades del país en conjunto con la Universidad de las Ciencias Informáticas y el Distrito Socialista Tecnológico en Mérida, Venezuela, están llevando a cabo el desarrollo del software SCADA y como parte de los módulos que componen el sistema está la Interfaz Hombre-Máquina (HMI). El módulo en la actualidad

se encuentra en desarrollo con algunos prototipos en funcionamiento pero con una serie de problemas que se exponen a continuación.

La biblioteca gráfica actualmente se encuentra empotrada en el módulo HMI ya que está diseñada específicamente para el SCADA en desarrollo, entre el componente de objetos gráficos y los componentes del HMI existe una marcada dependencia lo que no permite reutilizar los gráficos en aplicaciones similares limitando la reutilización e independencia de la biblioteca.

En algunas partes del módulo, el diseño arquitectónico resulta poco flexible y no permite la reutilización ya que no se ha tenido en cuenta la expansión de las funcionalidades del software y ejemplo de esto son los Sumarios del Sistema, de los cuales inicialmente se implementó el Sumario de Alarmas sin tener presente, que en total el número de sumario era mayor, por lo que los componentes que se desarrollaron para este sumario específico no pueden ser reutilizados ni están acorde a los requerimientos de los restantes por implementar.

Al módulo HMI con objetivos de pruebas se le implementó un sencillo generador de datos aleatorios para pruebas de concepto, pero para su puesta en funcionamiento necesita cambiar esta implementación y contar con un subsistema de Entrada/Salida robusto, que permita la comunicación con varios sistemas externos de forma concurrente y utilizando distintas vías de comunicación.

Como consecuencia del extenso período de trabajo, la implementación de gran cantidad de código al módulo HMI y al éste no contar con un código distribuido en la mayor cantidad posible de componentes independientes, los tiempos de compilación de la aplicación se han convertido en un verdadero problema para los desarrolladores provocando que éstos no puedan aprovechar el máximo la jornada laboral.

Por lo antes mencionado este trabajo de diploma se propone resolver el siguiente **Problema Científico** ¿Cómo lograr mejoras en el diseño arquitectónico del módulo HMI del SCADA?

El **Objeto de Estudio** de este trabajo es la interfaz hombre-máquina en procesos industriales y el **Campo de Acción** la arquitectura de la interfaz hombre-máquina en un SCADA.

De acuerdo con el Problema Científico planteado la **Idea a Defender** definida es la siguiente:

Si se aplican mejoras a la arquitectura del sistema actual, se obtendrán objetos gráficos reutilizables para cualquier aplicación, un diseño flexible que permite la integración de nuevos sumarios al SCADA y la comunicación de este módulo con otros del sistema por distintas vías.

De esta manera, se obtendrá un sistema más modular permitiendo mayor independencia entre los componentes.

El **Objetivo General** que se desea alcanzar es: Diseñar mejoras a la arquitectura del módulo HMI del SCADA.

Para llevar a cabo este trabajo y dar cumplimiento al objetivo propuesto se concibieron las siguientes **Tareas de Investigación**:

- Realización de un estudio de herramientas y tecnologías a utilizar para modelar el software.
- Definición de la estrategia de trabajo.
- Realización de un estudio de aplicaciones de código abierto con características arquitectónicas similares al Submódulo HMI para obtener diseños que puedan servir como referencia a la hora de tomar una decisión.
- Obtención de la ingeniería inversa a la actual implementación y realización de los diagramas de diseño, documentar y realizar el análisis de la arquitectura actual.
- Obtención de la documentación correspondiente a la arquitectura actual para exponer como está estructurado el sistema desde diferentes vistas arquitectónicas.
- Análisis crítico a la arquitectura para obtener un sumario de problemas por resolver.
- Estudio de los patrones de diseño y su aplicación a las posibles mejoras de la arquitectura del sistema para obtener diseños flexibles y reutilizables.
- Revisión de los requisitos no funcionales del SCADA que puedan tener repercusión en las propuestas de mejoras.
- Realización de los diagramas de diseño con las propuestas de mejoras para mostrar la solución.

Este documento está estructurado en tres capítulos, en los que se expone todo el proceso de desarrollo de este trabajo:

Capítulo 1: Se desarrolla la fundamentación teórica del presente trabajo.

Capitulo 2: Se realiza un análisis crítico del submódulo Ejecución de la Interfaz hombre-máquina.

Capitulo 3: Se describe, mediante diagramas las mejoras propuestas a la arquitectura actual.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En este capítulo se explican las generalidades de un sistema SCADA, el funcionamiento de éste y los principales módulos que lo componen. Se explica la necesidad del surgimiento de la Interfaz Hombre Máquina (HMI) y los beneficios que ésta proporciona. Además se hace una descripción de las principales tecnologías que se emplean en la modelación y construcción del software incluyendo la metodología de desarrollo, lenguajes de programación y modelado y las herramientas de desarrollo empleadas.

1.2 Generalidades de un sistema SCADA

Un sistema SCADA es una aplicación de software diseñada específicamente para funcionar sobre ordenadores que permiten supervisar y controlar a distancia una instalación, proceso o sistema de características variadas proporcionando comunicación entre los dispositivos de cuerpo, llamados también RTU (Remote Terminal Units o Unidades Remotas), donde se controla el proceso de forma automática desde la pantalla de una o varias computadoras.

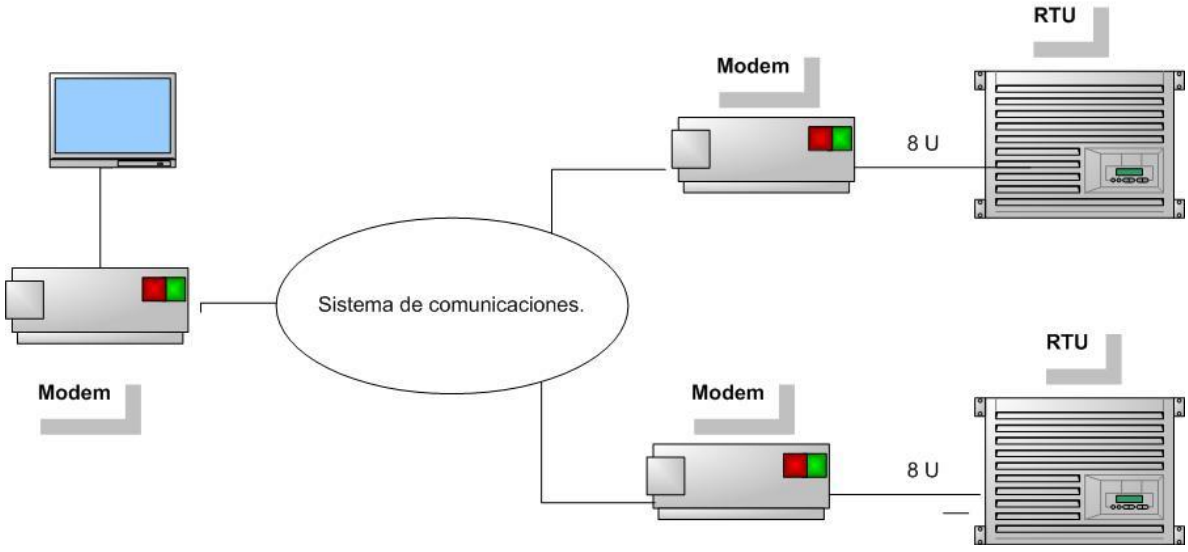

Fig. 1 Idea básica de un SCADA.

La estructura funcional de un sistema de visualización y adquisición de datos obedece generalmente a la estructura Maestro-Esclavo. La estación central (el maestro) se comunica con el resto de las estaciones (esclavos) requiriendo de éstas una serie de acciones y datos. [8]

El desarrollo de las computadoras ha permitido su implantación en todos los campos del conocimiento y a todos los niveles de aplicación.

Las primeras incursiones en el campo de la automatización localizaban todo el control en la PC y tendían progresivamente a la distribución del control en planta. De esta manera el sistema queda dividido en tres bloques principales:

- Software de control y adquisición de datos.
- Sistemas de adquisición y mando (sensores y actuadores).
- Sistema de interconexión (comunicaciones).

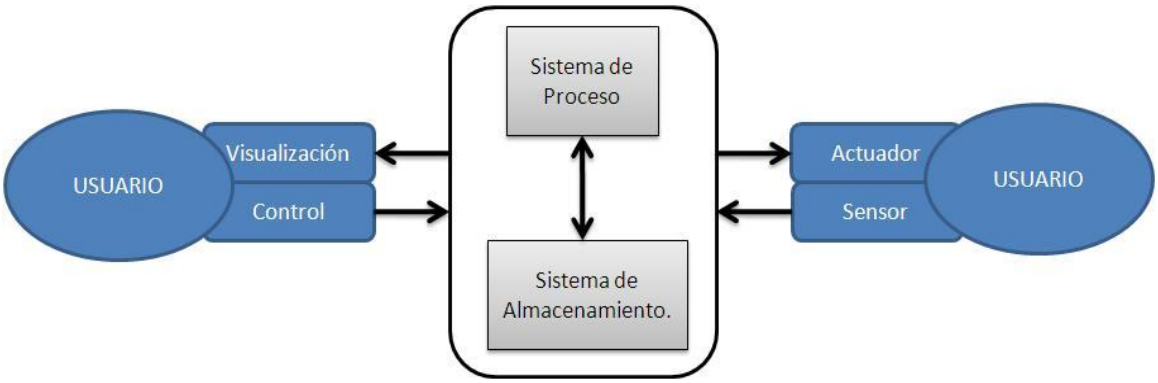

Fig. 2 Estructura básica de un sistema de supervisión y control.

El usuario mediante herramientas de visualización y control, tiene acceso al Sistema de Control de Proceso, generalmente una computadora donde reside la aplicación de control y supervisión (se trata de un sistema servidor). La comunicación entre estos dos sistemas se suele realizar a través de redes de comunicaciones corporativas (Ethernet).

El sistema de proceso capta el estado a través de los elementos sensores e informa al usuario mediante las herramientas HMI. Basándose en los comandos ejecutados por el usuario, el sistema de proceso inicia las acciones pertinentes para mantener el control del sistema mediante los elementos actuadores.

El software de control y adquisición de datos permite que el mundo de las máquinas se integre directamente a la red empresarial, pasando a formar parte de los elementos que permitirán crear las estrategias de empresas globales.

1.3 El módulo Interfaz Hombre-Máquina

La Interfaz Hombre-Máquina o HMI ("Human Machine Interface") es el aparato que presenta los datos a un operador (humano) y a través de la cual éste controla el proceso.

La industria de HMI nació esencialmente de la necesidad de estandarizar la manera de monitorear y de controlar múltiples sistemas remotos, PLC y otros mecanismos de control. Aunque un PLC realiza automáticamente un control pre-programado sobre un proceso, normalmente se distribuyen a lo largo de toda la planta, haciendo difícil recoger los datos de manera manual, los sistemas SCADA lo hacen de manera automática. Históricamente los PLC no tienen una manera estándar de presentar la información al operador. La obtención de los datos por el sistema SCADA parte desde el PLC o desde otros controladores y se realiza por medio de algún tipo de red, posteriormente esta información es combinada y formateada. Un HMI puede tener también vínculos con una base de datos para proporcionar las tendencias, los datos de diagnóstico y manejo de la información así como un cronograma de procedimientos de mantenimiento, información logística, esquemas detallados para un sensor o máquina en particular, incluso sistemas expertos con guía de resolución de problemas. Desde cerca de 1998, virtualmente todos los productores principales de PLC ofrecen integración con sistemas HMI/SCADA, muchos de ellos usan protocolos de comunicaciones abiertos y no propietarios. Numerosos paquetes de HMI/SCADA de terceros ofrecen compatibilidad incorporada con la mayoría de PLCs, incluyendo la entrada al mercado de ingenieros mecánicos, eléctricos y técnicos para configurar estas interfaces por sí mismos, sin la necesidad de un programa hecho a medida escrito por un desarrollador de software.

SCADA es popular debido a esta compatibilidad y seguridad. Ésta se usa desde aplicaciones pequeñas, como controladores de temperatura en un espacio, hasta aplicaciones muy grandes como el control de plantas nucleares. [14]

Un SCADA comprende los sinópticos de control y los sistemas de presentación gráfica. La función de un panel sinóptico es la de representar, de forma simplificada, el sistema bajo

control (un sistema de aprovisionamiento de agua, una red de distribución eléctrica, una refinería de petróleo).

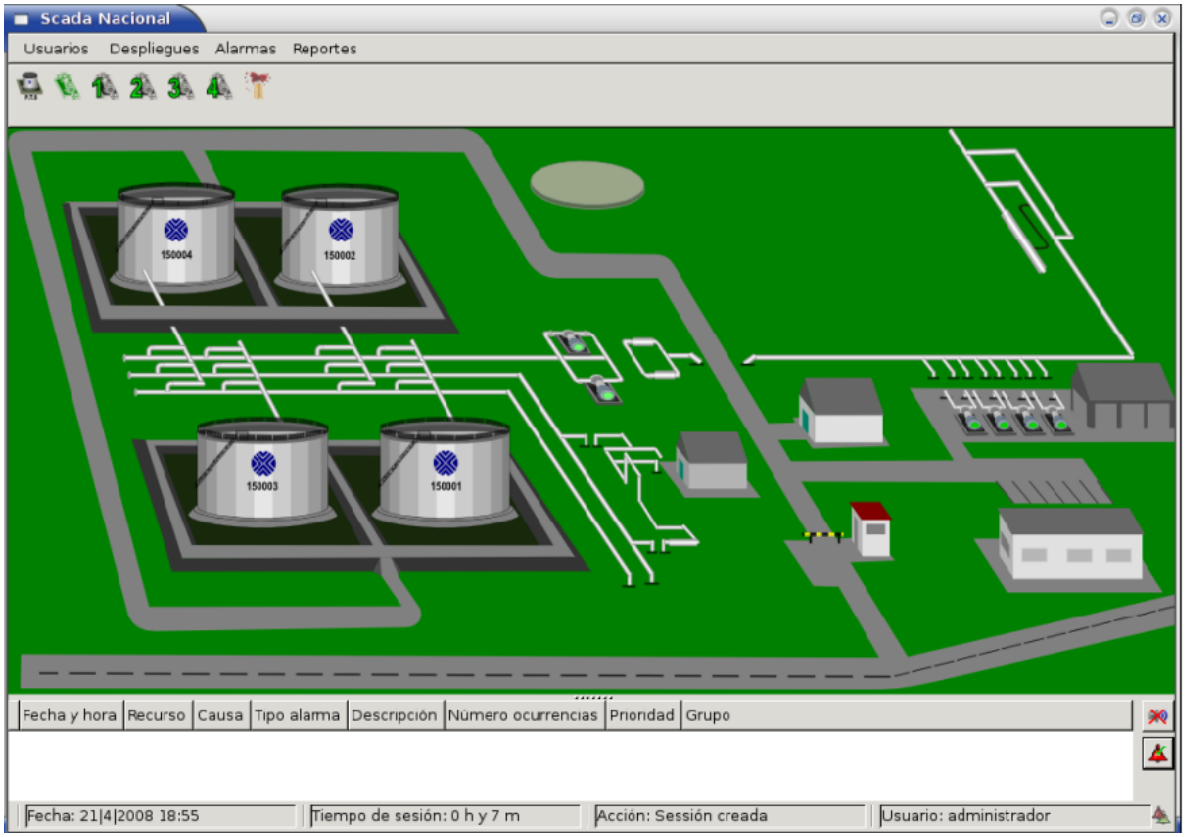

Fig. 3 Esquema de un sistema de llenado de tanques realizado con el prototipo HMI.

Como se observa, las HMI de hoy en día ofrecen a los operadores las más sofisticadas técnicas de supervisión y control pudiendo simular partes de la planta con los gráficos sinópticos haciendo sumamente intuitivo la operación de una planta específica.

1.4 Arquitectura de software

¿Qué es una arquitectura?

La arquitectura es lo que permite controlar el desarrollo del sistema desde la perspectiva técnica. La arquitectura de software se centra tanto en los elementos estructurales significativos del sistema (subsistemas, clases, componentes y nodos), así como en las colaboraciones que tienen lugar entre estos elementos a través de la interfaces.

Los casos de uso dirigen la arquitectura para hacer que el sistema proporcione la funcionalidad y uso deseados, alcanzando a la vez los objetivos de rendimiento razonables. Una arquitectura debe ser completa, pero también debe ser suficientemente flexible como para incorporar nuevas funciones, y debe soportar la reutilización de software existente.

¿Cómo se obtiene?

La arquitectura se desarrolla de forma iterativa durante la fase de elaboración pasando por los requisitos, el análisis, el diseño, la implementación y las pruebas. Utilizamos los casos de uso significativos para la arquitectura y un conjunto de otras entradas para implementar la línea base de la arquitectura, o "esqueleto" del sistema. Este conjunto de entradas incluye requisitos del software del sistema, middeware, que sistemas heredados deben utilizarse, requisitos no funcionales, y demás.

¿Cómo se describe?

La descripción de la arquitectura es una vista de los modelos del sistema, de los modelos casos de uso, análisis de diseño, implementación y despliegue. La descripción de la arquitectura describe las partes del sistema que es importante que comprendan todos los desarrolladores y otros interesados.

1.5 Tendencias y Tecnologías

En el presente epígrafe se realiza un análisis de las tendencias y tecnologías actuales en el campo del desarrollo de aplicaciones. Es importante destacar que cada uno de las herramientas o metodologías que se describen a continuación fueron escogidas por la Dirección del Proyecto SCADA Nacional.

Este proyecto se desarrolla a partir de las herramientas que brinda el software libre. Buscando la independencia tecnológica que estas brindan al posibilitar la libertad de uso y distribución de los programas sin incurrir en litigios de licenciamiento o asuntos legales.

1.5.1 Distribución de Linux: Debian GNU/Linux

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, es decir, sin un enfoque comercial, esto tiene ventajas y desventajas.

En primer lugar, las personas que se dedican a Debian tienen una alta motivación en participar en la misma, y se actualiza la distribución muy frecuentemente, apareciendo paquetes nuevos de software constantemente. Al mismo tiempo, existe un compromiso de calidad, no se desea distribuir software con errores.

En segundo lugar, dada su actitud abierta a la participación de todos, en el mismo espíritu original de Linux, constantemente hay personas que se unen a Debian para participar aportando su granito de arena, no solamente haciendo paquetes de programas, sino colaborando en el Servidor de Web, traduciendo documentación de Debian, documentando fallos, o ayudando a los usuarios a través de las listas de correo que mantiene la comunidad.

Como desventajas tiene un mayor componente técnico que otras distribuciones. También, es posible que ciertos paquetes no estén tan actualizados como debieran, quizás porque sus desarrolladores han dejado de actualizarlos y nadie se ha hecho cargo. Sin embargo esto es algo que todos los desarrolladores tratan de evitar y, aunque cada desarrollador mantiene sus paquetes, no es raro que otro desarrollador (incluso un usuario) envíe una nueva versión del paquete para arreglar un problema o actualizarlo. [19]

Se decidió usar la distribución Debian porque es una distribución de GNU/Linux de desarrollo muy estable, por lo que los paquetes que en esta se desarrollen 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).

1.5.2 Programación Orientada a Objetos.

La Programación Orientada a Objetos (POO u OOP según siglas en inglés) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (datos), comportamiento (procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

Según Grady Booch, es un "método de implantación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son miembros de una jerarquía de clases unidas mediante relaciones de herencia. En tales programas, las clases suelen verse como estáticas, mientras que los objetos suelen tener una naturaleza mucho más dinámica, promovida por la existencia de la ligadura dinámica y el polimorfismo". [18]

La programación orientada a objetos es una forma de programar. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Por la importancia y continua mención dentro en el documento se considera conveniente enunciar los más destacados:

- **Objeto:** Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad. Corresponden a los objetos reales del mundo que nos rodea, a objetos internos del sistema.
- **Clase:** Definiciones de las propiedades y comportamiento de un conjunto de objetos concretos. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.
- **Método:** Algoritmo asociado a un objeto, cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.
- **Evento:** Un suceso en el sistema. El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera.

- **Mensaje:** Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
- **Propiedad y atributo:** Contenedor de un tipo de datos asociados a un objeto, que hace los datos visibles desde fuera del objeto, y cuyo valor puede ser alterado por la ejecución de algún método.
- **Estado interno:** Es una propiedad invisible de los objetos, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto.
- **Componentes de un objeto:** Atributos, identidad, relaciones y métodos.
- **Representación de un objeto:** Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

Hay un cierto desacuerdo sobre exactamente qué características de un método de programación o lenguaje le definen como "orientado a objetos", pero hay un consenso general en que las características siguientes son las más importantes:

Abstracción: Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.
Encapsulamiento: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.
Principio de ocultación: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan

esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

Polimorfismo: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

Herencia: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que re-implementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple; esta característica no está soportada por algunos lenguajes (como Java).

1.5.3 Patrones.

Los patrones son una disciplina de resolución de problemas reciente en la ingeniería del software que ha emergido en mayor medida de la comunidad de orientación a objetos, aunque pueden ser aplicados en cualquier ámbito de la informática y las ciencias en general. Los patrones tienen raíces en muchas áreas, incluyendo la *literate-programming*.

De la misma forma que sucede con otros conceptos de la informática (y como es el caso de la arquitectura), no es fácil establecer una definición taxativa y definitiva del concepto de patrón.

En el libro “The Timeless Way of Building”, el arquitecto Christopher Alexander define al patrón en la siguiente manera:

“Cada patrón es una regla de 3 partes, que expresa una relación entre un contexto, un problema y una solución. Como un elemento en el mundo, cada patrón es una relación entre un contexto, un sistema de fuerzas que ocurren repetidamente en ese contexto y una configuración espacial que permite que esas fuerzas se resuelvan entre sí.”
“Como elemento de un lenguaje, un patrón es una instrucción que muestra como puede ser usada esta configuración espacial una y otra vez para resolver el sistema de fuerzas, siempre que el contexto lo haga relevante.” [19]

Se considera importante e interesante citar 2 principios postulados por Martin Fowler en “*Analysis Patterns: Reusable Object Models*” y que se debe tener en mente en todo momento al utilizar los patrones:

- Los patrones son un punto de partida, no un destino.
- Los modelos no están bien o mal, sino que son más o menos útiles. [16]

Documentar un patrón puede ser una tarea muy difícil. Citando a James Coplien en *Home of the Patterns Library*, un buen patrón:

Resuelve un problema: Los patrones capturan soluciones, no principios o estrategias abstractas.
Es un concepto probado: Capturan soluciones, no teorías o especulaciones.
La solución no es obvia: Muchas técnicas de resolución de problemas (como los paradigmas o métodos de diseños de software) intentan derivar soluciones desde principios básicos. Los mejores patrones generan una solución a un problema indirectamente (un enfoque necesario para los problemas de diseño más difíciles).
Describe una relación: Los patrones no describen módulos sino estructuras y mecanismos.
Tiene un componente humano significante: El software sirve a las personas. Los mejores patrones aplican a la estética y a las utilidades (de hecho, no es casual que varios de los primeros lenguajes de patrones tengan que ver con temas estéticos y utilidades). (Hillside Group, 2003)

1.5.4 Lenguaje de programación C++.

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C. Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes como ROOT (enlace externo).

Las principales características del C++ son el soporte para programación orientada a objetos y el soporte de plantillas o programación genérica (templates). Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores)
- Identificación de tipos en tiempo de ejecución.

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que dificulta mucho su aprendizaje. El nombre C++ fue propuesto por Rick Masciatti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

1.5.5 XML (E*x*tensible *M*arkup *L*anguage)

XML es una tecnología en realidad muy sencilla que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y con posibilidades mucho mayores.

XML, con todas las tecnologías relacionadas, representa una manera distinta de hacer las cosas, más avanzada, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. El XML juega un papel fundamental en este mundo actual, que tiende a la globalización y la

compatibilidad entre los sistemas, ya que es la tecnología que permitirá compartir la información de una manera segura, fiable y fácil. Además permite al programador y los soportes dedicar sus esfuerzos a las tareas importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de estos o el recorrido de las estructuras corre a cargo del lenguaje y está especificado por el estándar, de modo que el programador no tiene que preocuparse por ello.

1.5.6 Biblioteca gráfica GTK.

GTK+ (GIMP Toolkit) es una biblioteca que permite crear interfaces gráficas de usuario. Se distribuye bajo la licencia LGPL, por lo que posibilita el desarrollo de software libre, e incluso software comercial no libre que use GTK sin necesidad de pagar licencias o derechos.

GTK+ ofrece todo lo necesario para el desarrollo de interfaces gráficas, pasando por los «widgets» más básicos (botones, cajas de texto, menús, ventanas, etc) hasta otros mucho más complejos y elaborados que nos son de gran ayuda a la hora de programar aplicaciones gráficas. [7]

GTK+ está a su vez separado en varias librerías, algunas de ellas sólo disponibles para la versión 2.0:

- **GDK** implementa el nivel más bajo de la arquitectura, es decir, las primitivas gráficas. Es una librería que forma una capa sobre la implementación gráfica real (X Window, MS Windows, Mac OS X), y es por tanto la única parte de GTK+ que tiene que ser reescrita para soportar otra plataforma/sistema operativo. Es por esta razón por la que GTK+ ya ha sido portada a varios entornos (X Window, MS Windows, QNX, BeOS).
- **gdk-pixbuf** es la biblioteca que permite el tratamiento de imágenes gráficas. Esta librería permite el manejo (carga, visualización, grabación) de imágenes gráficas en distintos formatos (png, gif, jpeg, etc).
- Pango es un sistema que permite la representación de caracteres en distintos alfabetos (occidental, cirilico, árabe, chino, etc), y que supone uno de los pasos más importantes dentro del proyecto GNOME para la universalización del software libre.
- **ATK** es una librería de clases abstractas cuyo objetivo es servir de base para el desarrollo de aplicaciones accesibles para personas con deficiencias físicas. Es un

desarrollo de la empresa Sun, pues forma parte de su estrategia de inclusión de GNOME en entornos Solaris.

1.6 Metodología de software

1.6.1 Proceso Unificado de desarrollo (RUP):

El Proceso Racional Unificado o RUP (Rational Unified Process), es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El RUP es un producto de Rational (IBM). Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso).

El RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al terminar cada ciclo, estos se dividen en fases (inicio, elaboración, construcción y transición) que finalizan con un hito donde se debe tomar una decisión importante:

Se caracteriza por ser:

- **Dirigido por casos de uso**: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

- **Centrado en la arquitectura:** La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.

- **Iterativo e Incremental:** RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Por ejemplo, una iteración

de elaboración centra su atención en el análisis y diseño, aunque refina los requerimientos y obtiene un producto con un determinado nivel, pero que irá creciendo incrementalmente en cada iteración.

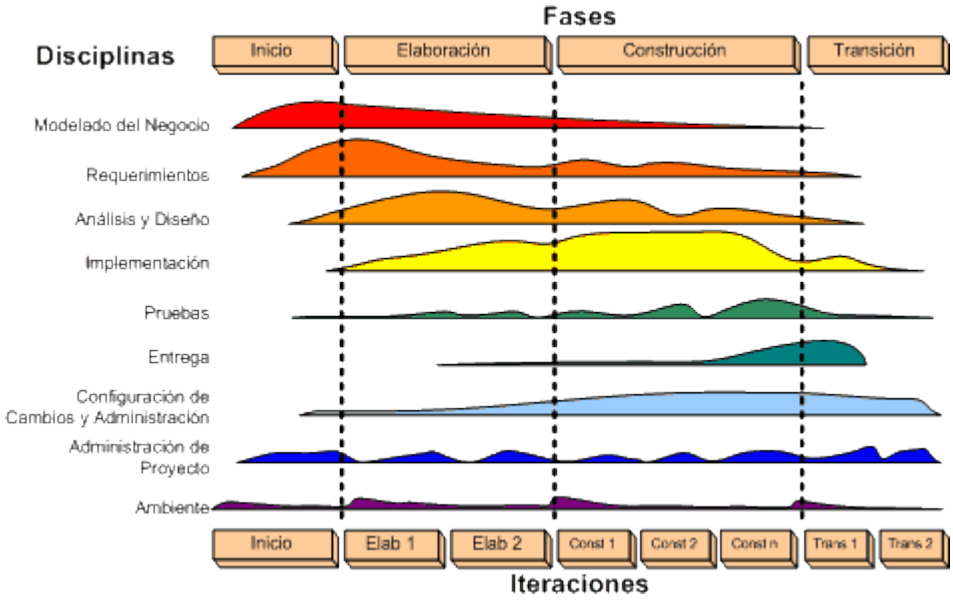

Fig. 4 Fases e iteraciones del Proceso Unificado.

UML:

Sus siglas en inglés (*Unified Modeling Language*), es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; 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. UML es un "lenguaje" para modelar y no un método o un proceso. [2]

1.7 Herramientas de desarrollo empleadas

1.7.1 IDE programación Eclipse

Eclipse es un entorno de desarrollo integrado (IDE) que abarca todo el ciclo de desarrollo de software. El software en sus inicios fue desarrollado por IBM pero en la actualidad es

mantenido por la Fundación Eclipse que es una organización independiente sin ánimos de lucro que fomenta una comunidad de código abierto y un conjunto de productos, capacidades y servicios complementarios además tiene el apoyo de gran cantidad de innovadoras empresas, prestigiosas universidades y varias personas alrededor del mundo. Inicialmente eclipse se desarrolló para los programadores que usaban el lenguaje Java pero en la actualidad existe soporte para varios lenguajes de programación como C/C++, Phyton, PHP entre muchos otros. También el software tiene herramientas para modelado de software, sistemas de gestión de bases de datos (DBMS), para la gestión de la configuración y el control de versiones tiene soporte para CVS y Subversion, existen plugins para la realización de pruebas de unidad como (JUnits, CxxTest). La versión utilizada en el proyecto es la 3.3 (Europa). [12]

1.7.2 Herramientas Case

Las herramientas CASE (***C**omputer **A**ided **S**oftware **E**ngineering*, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar a ingenieros, desarrolladores en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, realización de ingeniería inversa entre otras funciones. Para el desarrollo del producto de software se utiliza la herramienta **IBM Rational Software Architect (RSA) Versión 7.0.0.2.** RSA una herramienta comercial que fortalece el desarrollo dirigido por modelos con UML para la creación de servicios y aplicaciones con arquitecturas sólidas.

Rational Software Architect permite unificar todos los aspectos del desarrollo y el diseño de software en una sola herramienta. Refuerza la plataforma de modelado ampliable y abierto ya que está basada en Eclipse. Permite la codificación y modelado para Java/J2EE, Web Services, SOA y aplicaciones escritas en C++. Posibilita administración flexible de modelos para el desarrollo paralelo y modificación de la arquitectura permitiendo dividir, combinar, comparar y mezclar modelos y fragmentos de modelos. Permite la generación de código a partir del modelo y viceversa eliminando tareas repetitivas como la generación de código. Permite la generación de reportes en formatos

HTML, PDF y XML de los diseños UML construidos Está soportada para los sistemas operativos Linux, Windows 2000, Windows Server 2003, Windows XP. Es una herramienta utilizada en el desarrollo de software por decisión de la dirección del proyecto. [13]

Capítulo 2: Análisis crítico a la arquitectura del HMI Ejecución

2.1 Introducción

En el presente capítulo se hace un análisis de la arquitectura primeramente haciendo una descripción general de su diseño para luego hacer referencia al estilo arquitectónico utilizado y a los patrones de diseño. Se exponen los requerimientos no funcionales que puedan afectar el diseño del software desde el punto de vista arquitectónico. Luego se hace un análisis desde un punto de vista crítico de todas las partes de la arquitectura que están afectando el desarrollo actual del módulo.

Se presentan los diagramas obtenidos a través de la técnica de Ingeniería Inversa con la cual se trata de analizar el sistema para reconstruir la descripción de alto nivel a partir de la de bajo nivel. Básicamente consiste en analizar el código para extraer relaciones y el diseño de todo o parte del sistema. Con la ingeniería inversa se trata de re-documentar el sistema y descubrir información de diseño con el fin de poder entenderlo mejor.

La reconstrucción de la arquitectura resulta una representación arquitectural que puede:

- Ser usada para documentar la arquitectura existente.
- Ser usada para chequear la conformidad con la ya implementada a la arquitectura diseñada.
- Servir como un punto de partida para aplicar reingeniería al sistema para diseñar una nueva arquitectura a través de la estrategia de transformación de la arquitectura.
- Ser usada para identificar componentes para establecer un método de línea de aplicación.

2.2 Descripción General de la arquitectura del sistema.

Una vez que se determinan los principales factores que permiten obtener una arquitectura adecuada, organizativa y un análisis profundo del sistema; más la información obtenida en la primera fase, realizada por el analista del sistema, la persona o el equipo que define la arquitectura obtiene una primera versión de lo que es la línea base de la arquitectura. La línea base de la arquitectura es una versión interna, que está centrada en la descripción de la arquitectura, y serviría como estrategia de trabajo a seguir.

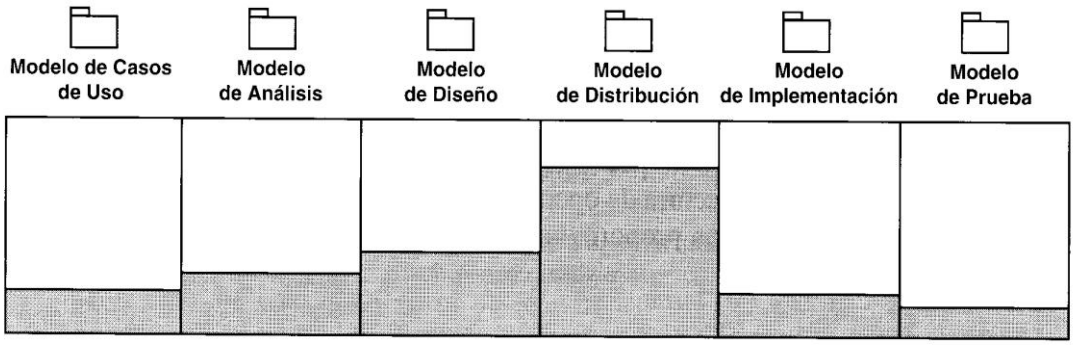

Figura 5 La línea base de la arquitectura es una versión interna del sistema, que está centrada en la descripción de la arquitectura.

2.3 Estilo arquitectónico que se utiliza en el Sistema.

La arquitectura actual persigue que el módulo HMI pueda dar respuesta a requisitos de portabilidad, comprensibilidad, extensibilidad y eficiencia. Uno de los requisitos que más estresan la solución es la cualidad de que el sistema pueda desplegarse en ambiente consola tipo escritorio y en la Web. Por un lado resulta deseable el desempeño que se logra con las aplicaciones nativas en modo consola gráfica, pero se desea tener la facilidad y portabilidad que brindan las tecnologías Web. En esta primera etapa del desarrollo no se aborda la capacidad de la herramienta de ser visualizada sobre la Web. No obstante la arquitectura propuesta debe garantizar que la futura incorporación de estas cualidades represente el mínimo en costo en cuanto a esfuerzo se refiere.

La solución está orientada al patrón arquitectónico Modelo-Vista-Controlador (MVC), el cual es ampliamente utilizado por soluciones computacionales afines al área objeto. Es de destacar que el patrón MVC no se aplica de forma tradicional donde son identificables los paquetes correspondientes al Modelo, la Vista o el Control. Ésto se deriva de los escenarios donde se requiere sea desplegada la aplicación. Por ejemplo, en una buena proporción de las aplicaciones Web es fácil de identificar las partes como la Vista y el Control donde todas las acciones de los usuarios del sistema son convertidas en pedidos HTTP que llegan al servidor donde pueden ser atendidos por el componente Control de la arquitectura, luego, cuando se tienen los datos listos para la salida hacia el cliente Web se le pasa el control al componente Vista para que le proporcione formato a esa información produciendo el HTML final que llega al usuario.

Esta situación contrasta con el escenario de una aplicación de escritorio donde la filosofía de las bibliotecas gráficas busca que cada componente sea la unidad responsable de la captura de los eventos que genera el usuario (equivalentes al Control) que además tienen también la responsabilidad de representar la información a través de su área de dibujo.

En este contexto es común enlazar el Modelo a través de callbacks con el componente y a partir de éstos, generar las actualizaciones de la vista. La asimetría de las implementaciones hace necesario establecer un conjunto de abstracciones que permitan generalizar ambos enfoques para poder dar solución a los requisitos de portabilidad de una forma lo más natural posible. El primer paso en esa dirección corresponde a la identificación y selección de conceptos que no interceptan las funcionalidades de Vista y Control. Estos conceptos serán agrupados en lo que se denominará el subsistema Modelo por lo que serán reutilizables tanto para el escenario Web como para el de escritorio. Por otra parte las funcionalidades que semánticamente involucren lógica de Vista o de Control serán implementadas particularmente para cada escenario.

El subsistema de visualización del SCADA Nacional se divide en dos aplicaciones fundamentales:

1. Ambiente de Ejecución (AE).
2. Ambiente de Configuración (AC).

El AE es la aplicación encargada de brindar la interfaz hombre-máquina (HMI) para monitorear y supervisar los procesos y el AC es la aplicación que posibilita la configuración de la mayoría de los subsistemas del SCADA incluyendo el AE.

Es importante destacar que existe una relación de extensión entre ambos ambientes, debido a que comparten funcionalidades importantes, como las relacionadas con los conceptos de Despliegue, Menús, un conjunto de objetos gráficos, entre otros. Teniendo en cuenta esta relación se diseñó una estrategia de desarrollo que permitiera la reutilización de los elementos comunes para evitar re-trabajo.

Modelo Vista Controlador (MVC): el patrón fue descrito por primera vez en 1979 por Trygve ReenksKaug, entonces trabajando para Smalltalk en laboratorios de investigación de Xerox. La implementación a fondo del patrón está descrita en "*Programación de Aplicaciones en Smalltalk-80(TM): Como utilizar Modelo Vista Controlador*".

Descripción del patrón (MVC).

- **Modelo:** Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de éstos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo o calculando si hoy es el cumpleaños del usuario.
- **Vista:** Presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- **Controlador:** Éste responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

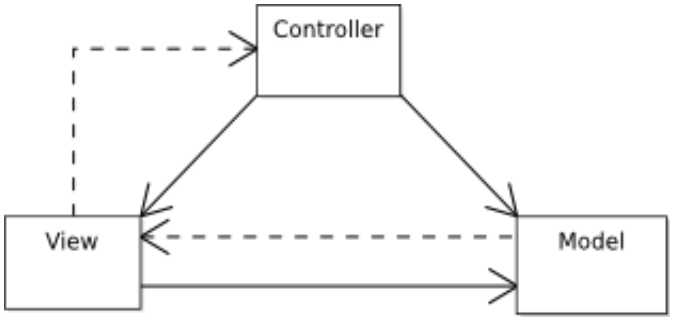

Fig. 5 Diagrama Modelo Vista Controlador. [4]

El diagrama del la figura 4 muestra la relación entre el modelo, la vista y el controlador. Las líneas sólidas indican una asociación directa, y las punteadas una indirecta (Que en ocasiones en la aplicación se implementa con el patrón Observer.)

Patrones generales de software para asignar responsabilidades (GRASP) [1].

Patrón	Descripción
Experto	¿Quién asumirá la responsabilidad en el caso general? Asignar una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad.
Creador	¿Quién crea? Asignar a la clase B la responsabilidad de crear una instancia de la clase A, si se cumple una de las siguientes condiciones. 1. B contiene A 2. B agrega A 3. B tiene los datos inicializados de A 4. B registra A 5. B utiliza A muy de cerca
Controlador	¿Quién administra un evento del sistema? Asignar la responsabilidad de administrar un mensaje de eventos del sistema una clase que representa una de las siguientes opciones. 1. El negocio o la organización global. 2. El sistema global (Un controlador de fachada). 3. Un ser animado del dominio que realice el trabajo (Un controlador de papeles). 4. Una clase artificial (Fabricación Pura) que represente el caso de uso (Un controlador de casos de uso).
Bajo Acoplamiento	¿Cómo dar soporte a poca dependencia y una mayor reutilización? Asignar responsabilidades de modo que se mantenga el bajo acoplamiento.
Alta cohesión	¿Cómo mantener controlable la complejidad? Asignar las responsabilidades de modo que se mantenga una alta cohesión.
Polimorfismo	¿Quién cuando el comportamiento varía de algún tipo? Cuando varía el tipo (clase) de alternativas o comportamientos relacionados, asignar la responsabilidad a una clase artificial que no represente nada en el dominio del problema, a fin de brindar soporte a una alta cohesión, a bajo acoplamiento y a la reutilización.
Fabricación Pura	¿Quién, cuando uno está desesperado y no quiere violar los patrones de Alta Cohesión ni Bajo Acoplamiento?

	Asignar un conjunto muy alto de responsabilidades a una clase artificial que no represente nada en el dominio del problema, a fin de brindar soporte a una alta cohesión, a bajo acoplamiento y a la reutilización.
Indirección.	¿Quién para evitar el acoplamiento directo? Asignar la responsabilidad a un objeto intermedio para que medie entre otros componentes o servicios, de modo que no se acoplen directamente.
No Hables con Extraños (ley de Demeter)	¿Quién para no conocer la estructura de los objetos indirectos? Asignar la responsabilidad al objeto directo del cliente para colaborar con el objeto indirecto, de modo que el cliente no necesita conocer el objeto indirecto. En el método, los mensajes únicamente pueden enviarse a los siguientes objetos. • El objeto **this** (o el **self**). • Un parámetro del método. • Un atributo de **self**. • Un elemento de una colección que sea un atributo de **self**. • Un objeto creado dentro del método.

Tabla 2.2. Patrones generales de software para asignar responsabilidades (GRASP) [1].

2.4 Requerimientos no funcionales.

En este epígrafe se especifican los requerimientos no funcionales que se tuvieron en cuenta para el diseño y desarrollo de los prototipos del sistema.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe cumplir. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido y confiable.

Requerimientos de Rendimiento y Disponibilidad del sistema.

- Se debe garantizar el tiempo de acceso a las diferentes ventanas y despliegues del sistema en menos de dos segundos.
- Se debe garantizar que las modificaciones realizadas sobre la configuración del sistema (datos, despliegues, entre otros) se envíen al registro de eventos y se reflejen en todos los nodos y sistemas asociados en un tiempo de un segundo.

- Se debe garantizar el soportar a una configuración de redundancia de redes, de forma tal que pueda restablecer conexión a alta velocidad en todos los nodos asociados (consolas, servidores en cluster, interfaces con otros sistemas).

Requerimientos de Disponibilidad del sistema.

- Se debe confeccionar un manual de usuario.
- Se debe confeccionar un curso o guía de adiestramiento para cada perfil de usuario en el sistema.

Requerimientos de portabilidad.

- El sistema debe funcionar en sistemas de la familia GNU/Linux.

Requerimiento de Confiabilidad

- Se debe garantizar que el módulo esté disponible las 24 horas del día.
- Las actividades de mantenimiento, supervisión y edición del sistema no deben interferir en el correcto desempeño del resto del sistema, ni afectar la ejecución de la aplicación.

Requerimientos del tiempo de respuesta para adquisición y procesamiento de datos.

- 5 milisegundos para acceso a los datos.
- 1-2 segundos para el refrescamiento de los datos en el despliegue de las consolas. Tiempo que demora el dato desde que sale de módulo de recolección hasta que se visualiza en la consola.
- 5-10 segundos para las consultas a base de datos utilizando sentencias SQL.
- 1 segundo para la propagación de los datos al resto de los nodos y sistemas asociados. Tiempo en que los datos llegan a los nodos redundantes.

2.5 Representación de la arquitectura desde un punto de vista crítico.

2.5.1 Descripción de la línea base:

Según RUP, cada ciclo de trabajo produce una *versión* del Sistema, y esta es un producto que incluye los artefactos que se crean durante la vida del proyecto, dentro de estos artefactos están los diferentes modelos que permitirán la representación de la descripción de la arquitectura.

La descripción de la arquitectura se obtiene de dichas versiones, además es un extracto o un conjunto de vistas de los modelos que están en la línea base de arquitectura. A continuación se explicará cómo se representa la arquitectura.

¿Cómo se representaría la arquitectura del Sistema?

Para la representación de la arquitectura se definió como meta un conjunto de objetivos como la reutilización de partes del sistema en una nueva aplicación, adicionar funcionalidades y dividir la solución en la mayor cantidad de componentes posibles. Para comprender la arquitectura se generarán las vistas:

Vista de casos de uso: Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los *casos de usos* o escenarios del modelo de casos de uso más significativos, con las funcionalidades centrales del sistema. Si el sistema se hace extenso entonces se debería organizar en paquetes, lo cual facilitaría la comprensión de la vista de casos de uso.

Vista lógica: Esta vista representa un subconjunto del artefacto Modelo de diseño, la cual representa los elementos de diseño más importantes para la arquitectura del sistema. Éste describe las clases fundamentales, su organización en paquetes y subsistemas. También describe las realizaciones de casos de uso más importantes como por ejemplo las que describen aspectos dinámicos del sistema.

Vista de implementación: Esta vista describe la descomposición del software en capas y subsistemas de implementación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

2.5.2 Vista de Casos de Usos.

En esta vista se presentan los casos de uso y actores que afectan a la Arquitectura del Sistema. El modelo de casos de uso del sistema sirve como acuerdo entre clientes y desarrolladores, y proporciona la entrada fundamental para el análisis, el diseño y las pruebas. Un modelo de casos de uso del sistema incluye los casos de uso del sistema y los actores del sistema.

En el modelo de casos de uso del sistema los actores representan terceros fuera del sistema que colaboran con este. Cada forma en la que los actores usan el sistema se representa como casos de uso. Los casos de uso son fragmento de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. Un caso de uso especifica una secuencia de acciones que el sistema debe llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia.

Actores	Justificación
Operador	Persona encargada directamente del control y supervisión de los procesos operacionales y de la ejecución y envío de los reportes durante su guardia. El ambiente del Operador es la Sala de Control, este es el responsable de las consolas de operación, que están conectadas a los servidores SCADA, los cuales pueden encontrarse fuera de la sala de control.
Mantenedor	Persona encargada de configurar el sistema SCADA con las características del proceso y del negocio, para que pueda ser empleado por el operador y los demás involucrados. Se encuentra adyacente a la sala de control y/o de los servidores del SCADA. Puede visualizar las operaciones y despliegues de los procesos supervisados, además de preparar y editar todo el ambiente requerido por el operador.

Tabla 2.3. Actores del sistema.

Diagrama de Casos de Uso.

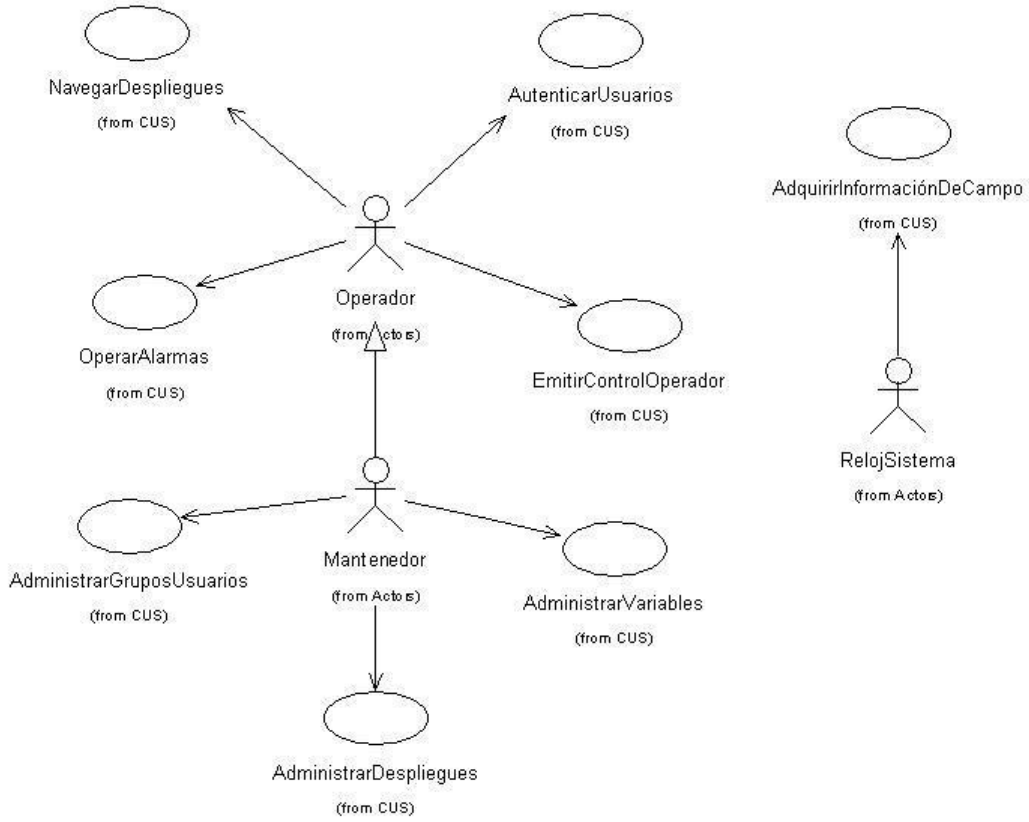

Fig. 6 Diagrama de Casos de Uso significativos para la arquitectura del sistema

2.5.3 Vista Lógica.

Esta vista describe las partes más significativas de la arquitectura en un modelo de diseño, así como su descomposición en subsistemas y módulos contenidos en esta vista y para cada subsistema significativo su descomposición en clases. Introduce las clases arquitectónicamente significativas y la descripción de sus responsabilidades así como las relaciones más importantes, operaciones y sus atributos. Solo se representarán las partes que están relacionadas con los objetivos de reingeniería planteados.

Descripción Global.

La siguiente figura representa la división del sistema en subsistemas, esto se hace con el objetivo de hacer que el mismo sea más re-usable y facilite el mantenimiento.

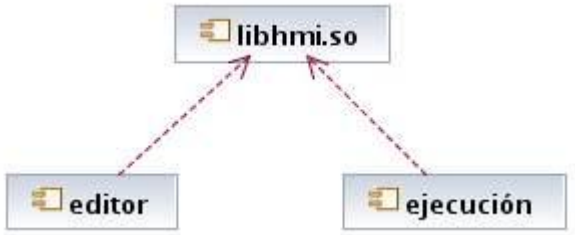

Fig. 7. Diagrama de componentes arquitectura actual.

- El ambiente de ejecución (AE) es la aplicación encargada de brindar la interfaz hombre-máquina (HMI) para monitorear y supervisar los procesos
- El editor (AC) es la aplicación que posibilita la configuración de la mayoría de los subsistemas del SCADA incluyendo el AE.

Como ya se explicó en una sección anterior, el componente editor y runtime comparten varias funcionalidades comunes. El componente libhmi.so contiene todas las características comunes de ambos sub-módulos. Al comienzo se pensó que solo con estos componentes sería suficiente en el sistema. Hoy en día tener solo 3 componentes con la extensa cantidad de código fuente por la que está compuesta cada uno, provoca que la compilación sea un proceso lento y el equipo de desarrolladores no aproveche al máximo la jornada laboral por las esperas que provoca ver el resultado de un pequeño cambio en la implementación. Otro problema es la necesidad de trabajar con todo el código fuente de la aplicación por la dependencia entre clases que generan un componente y otro. Esto provoca que los desarrolladores no puedan construir y probar componentes de forma independiente.

Paquetes de diseño del módulo.

Los paquetes de diseño del módulo describen la descomposición global o general en que se dividirá el modelo de diseño, en términos de paquetes, dependencias y capas. Como se puede observar las dependencias entre paquetes es bastante compleja, cada uno de estos se compila independientemente pero por el gran número de dependencias de

clases de un paquete y otro se hace demasiado complejo haciendo casi imposible determinar el orden de generación de los distintos componentes. Tener los paquetes tan relacionados provoca un alto nivel de complejidad a la hora de separar la aplicación en componentes independientes. Como la aplicación está dividida por paquetes pero no en la mayor cantidad de componentes posibles conlleva a que sea necesario poseer todo el código de la aplicación y no poder desarrollar componentes separados.

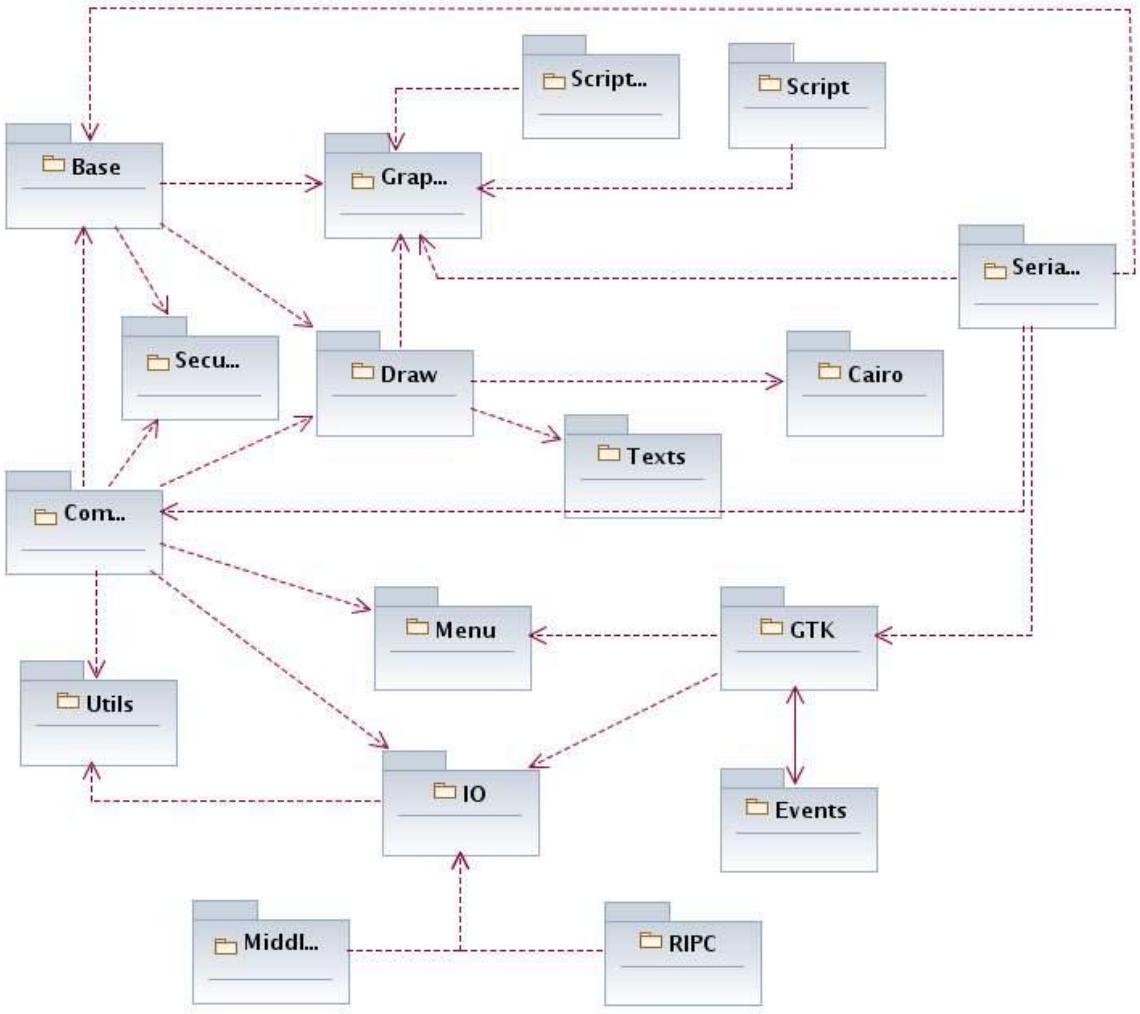

Fig. 8 Vista de paquetes.

Diagrama de clases de los widgets del SCADA.

En el diagrama se definen las principales interfaces de los objetos gráficos que se muestran en los distintos despliegues. A partir de estos objetos e interfaces se construyen los objetos que se representan en el visualizador como pueden ser cajas de texto (Edit), imágenes, botones, etc. En el diagrama se pone de ejemplo un objeto de tipo Edit el cual tiene como clase base a Edit de **Graphics** y concreto Edit de **Draw** que define las características y la forma en que se pinta en pantalla este objeto. Como se puede observar se heredan características propias para el visualizador del SCADA desde el Edit de **Graphics** como es la característica de ser actualizable y lo mismo pasa para el resto de los objetos gráficos provocando que estos objetos no se puedan reutilizar en aplicaciones que no necesiten ser actualizables como una aplicación multimedia, un editor de gráficos, etc.

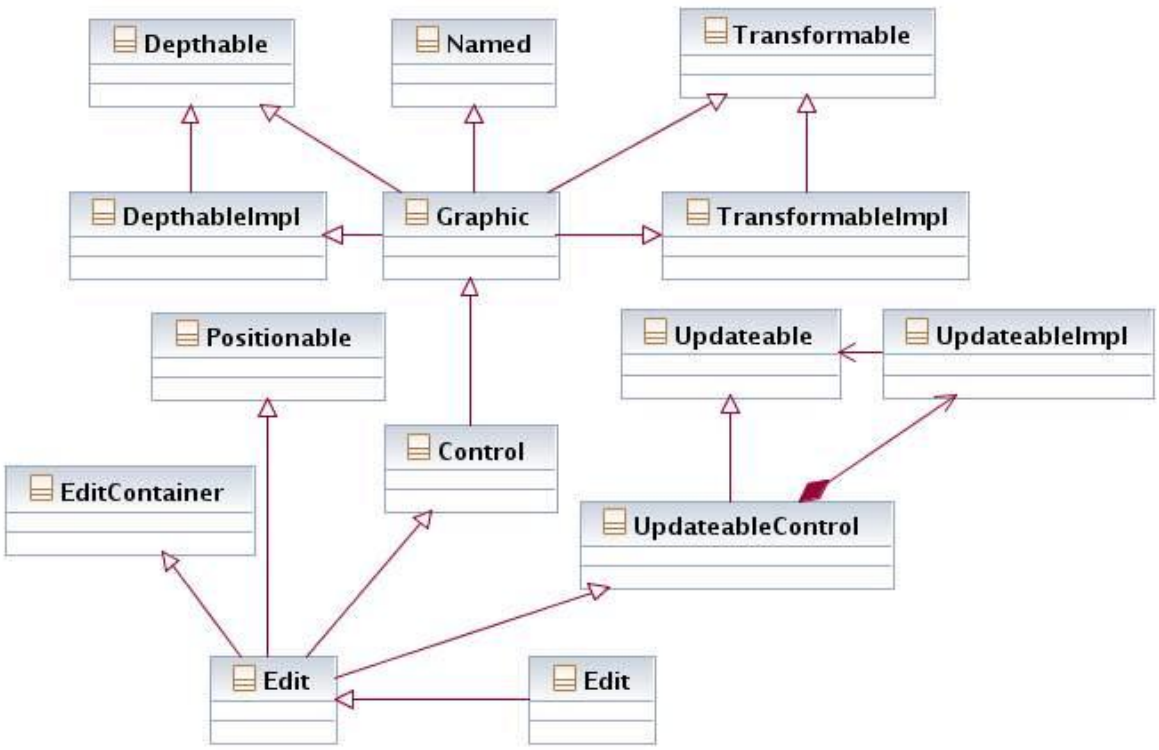

Fig. 9 Diagrama de clases de los widgets.

Subsistema Entrada/Salida.

En el diagrama se muestra el subsistema Entrada/Salida existente en el módulo de ejecución el cual tiene que permitir:

- Implementar un mecanismo de comunicación con otros módulos utilizando la capa middleware diseñada específicamente para el SCADA.
- Comunicación con varios módulos de forma simultánea.
- La comunicación puede ser síncrona o asíncrona lo que no puede tener gran impacto en el diseño del sistema Entrada/Salida.
- Envío y recepción de comandos hacia otros módulos.
- Recepción de variables en tiempo real provenientes de una o varias fuentes de datos.
- Recepción de alarmas que llegan al HMI y almacenar en memoria las alarmas que se mantienen activas.
- Recepción de los estado de comunicación de los diferentes dispositivos y mantener en memoria estos estados.
- Encuestar a la Base de Datos de Históricos y proveer de datos a los componentes que lo soliciten.
- La implementación se debe abstraer del mecanismo utilizado para obtener y enviar datos.

Como se observa el diagrama de Entrada/Salida es un mecanismo sencillo y no permite ninguna de las funcionalidades expuestas anteriormente. Se ve la necesidad de rediseñarlo para cumplir con los requerimientos especificados.

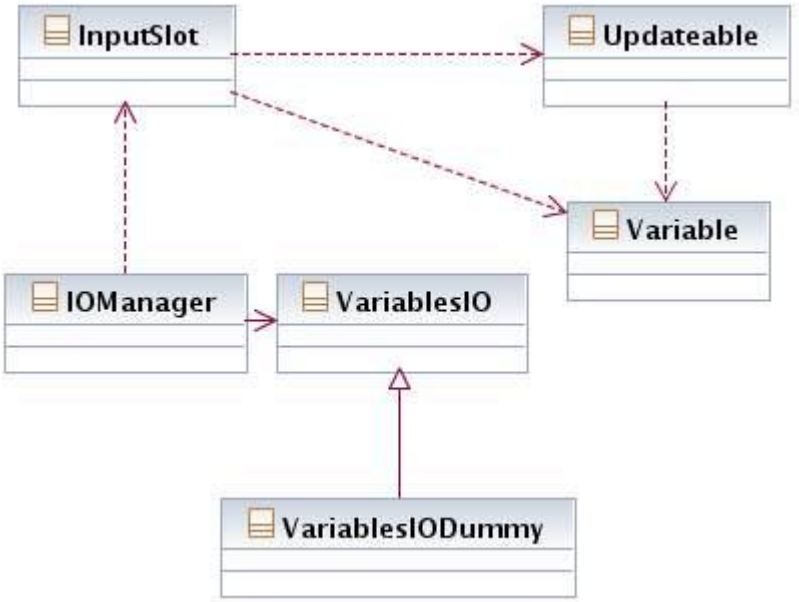

Fig. 10 Subsistema Entrada/Salida HMI Ejecución.

Subsistema Sumarios del SCADA.

Un Sumario del SCADA se refiere a un despliegue tipo ventana emergente que permite obtener información detallada sobre un determinado recurso del Sistema, como por ejemplo de puntos, dispositivos y subcanales. Es una interfaz gráfica que concentra las condiciones particulares de la información visualizada sin referencia grafica del proceso asociado. Los sumarios con que debe contar el módulo HMI son:

Diagrama de clases Sumario de Alarmas.

En el siguiente diagrama se pueden observar las clases correspondiente al Sumario de Alarmas que fue el primero en diseñarse al sistema, en el diseño que se tiene no existe la posibilidad de adicionar nuevas funcionalidades reutilizando las comunes que tienen todos los sumarios del sistema por lo que surge la necesidad del rediseño de este teniendo en cuenta los sumarios adicionales que se exigen y posibles incorporaciones de nuevas funcionalidades.

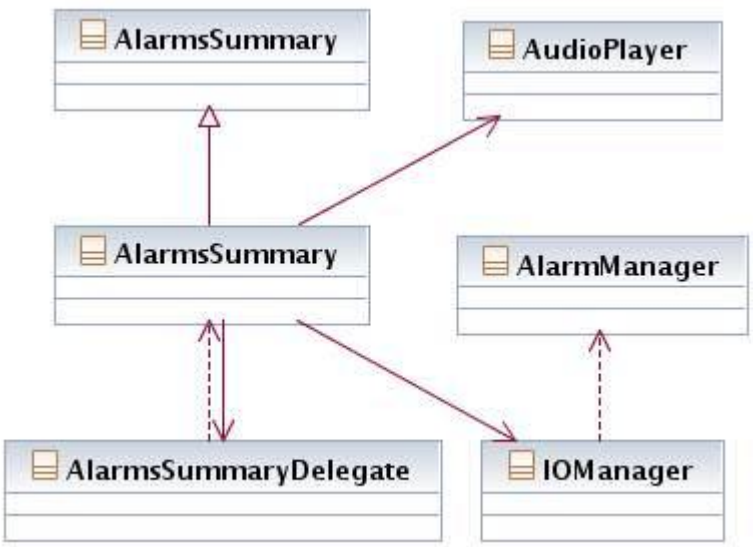

Fig. 11 Subsistema Sumarios del SCADA.

Capítulo 3: Descripción de la Solución Propuesta.

3.1 Introducción

En este capítulo se hace una descripción de las distintas soluciones que se le dan a los problemas descritos en el capítulo 2 y las ventajas que éstas implican sobre el diseño con que anteriormente se contaba. De forma general se le da solución a los problemas de reutilización de componentes, se plantea un diagrama de componentes con la aplicación lo más modular posible y se modifica el diseño de algunas partes del sistema para dar cabida a nuevas funcionalidades que no eran soportadas con el diseño que anteriormente se contaba.

3.2 Consideraciones generales acerca del diseño.

La propuesta de solución está diseñada basada en los estándares utilizados en la arquitectura previamente analizada que incluye estándar de codificación, programación orientada a objetos, y la aplicación de distintos patrones como alta cohesión, bajo acoplamiento, creador, fábrica, polimorfismo entre otros muchos más que posibilitan diseños claros y flexibles con posibilidades de incluir mejoras y funcionalidades con el mínimo esfuerzo. Como uno de los principales principios a tener en cuenta está la utilización del patrón arquitectónico MVC.

3.3 Descripción de la solución

Al comienzo del presente documento se hizo referencia a una serie de inconformidades que se tenían con la actual arquitectura del sistema y en el capítulo 2 mediante un análisis de la implementación del sistema se profundizó desde el punto de vista técnico y se plantearon posibles causas. En esta sección se tratará de dar solución a los problemas existentes.

A continuación se hace referencia a los problemas y las soluciones propuestas.

3.3.1 Lenta compilación del código fuente.

El módulo HMI actual tiene como uno de los problemas el tiempo de compilación que a medida que se le van adicionando clases y dependencias de otras bibliotecas se hace cada vez más lento construir un ejecutable. En el estado actual del proyecto la misma puede llegar a tardar más de una hora por lo que se hace necesario buscar soluciones que permitan optimizar el proceso.

La solución que se propone es dividir la aplicación en 5 componentes, éstos serían independientes de tal forma que una vez compilado un componente mientras no se modifique el código asociado a éste el compilador no tendrá que construirlo, solo al final se vincularía para crear el binario con todos los objetos compartidos de los que se tenga dependencia. A continuación se muestra como debe quedar el diagrama de componentes de la propuesta:

Descripción del diagrama de componentes propuesto.

Ventajas sobre el diagrama de componentes anterior:

- Las funcionalidades comunes entre los módulos “HMI Editor” y “HMI Ejecución” quedan implementadas en las clases que genera el objeto compartido “HMIBase.so” lo que permite que una modificación en uno de los tres componentes no implique la compilación de todo el código, solo el componente afectado.
- La existencia de una biblioteca grafica “Widgets.so” que no depende directamente del código de la aplicación posibilita la reutilización de este componente en aplicaciones similares y elimina también la necesidad de recompilar el código de la biblioteca junto con la aplicación principal.
- La creación del componente “IO.so” permite que se puedan crear ejecutables que no tengan que depender de “HMI Ejecución” sino que pueden trabajar de forma independiente del programa principal y es el caso de poder crear los sumarios de alarmas “SystemSummary” independientes posibilitando tener menos clases en la aplicación principal y haciendo más modular el software.
- Una de las causas de la gran demora en la compilación y gran consumo de memoria RAM era el uso de serialización para persistir los objetos en el disco duro. El problema reside en que muchas clases tienen que serializarse y cuando esto se hace sobre un mismo binario todas lo hacen a la vez por lo que agota la

memoria de la PC y provoca grandes atrasos a los desarrolladores. Con la nueva propuesta los objetos a serializar estarían divididos por los distintos componentes por lo que eliminaría en parte el problema existente.

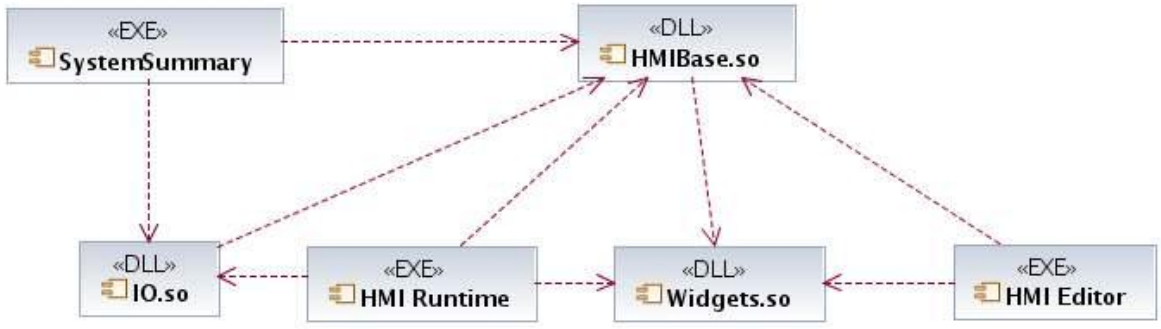

Fig. 12 Diagrama de componentes propuesto.

3.3.2 Necesidad de independizar la biblioteca gráfica del HMI.

Se propone adicionar otro nivel a la jerarquía de objetos gráficos, en este nivel se contendrán todas las características básicas de widgets genéricos de forma tal que se pueda incluir la biblioteca existente en aplicaciones de propósito general y construir una capa concreta para el SCADA que proporcione capacidades personalizadas para software de este perfil. El siguiente diagrama muestra un ejemplo de cómo quedaría la nueva jerarquía de clases.

Descripción del diagrama de clases propuesto.

En el diagrama se puede observar la aparición de otro nivel concreto “Updateable” en la jerarquía de objetos el cual implementaría solo las características correspondientes a los objetos de un SCADA. En las capas Draw y Graphics quedaría la responsabilidad de características comunes en este ejemplo de un “Edit” que serían las de mostrar información y permitir modificar el valor que se le asocie al componente.

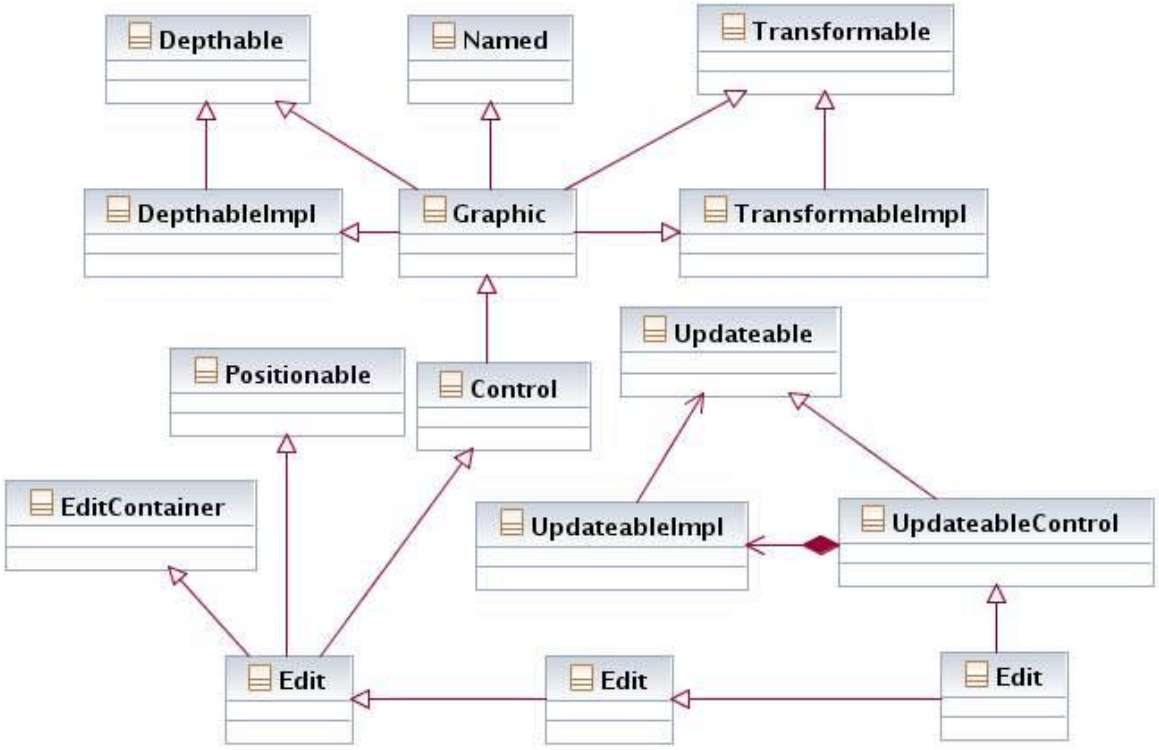

Fig. 13 Diagrama propuesto con la nueva abstracción de los widgets.

3.3.3 Subsistema Entrada/Salida.

Con el análisis expuesto en el capítulo anterior, las funcionalidades a adicionar al sistema provocan que se tenga que rediseñar el sistema de Entrada/Salida para el submódulo de Ejecución.

Descripción de la solución propuesta.

A continuación mediante distintos diagramas se muestra como queda la solución planteada.

Diagrama General del Mecanismo Entrada/Salida.

Este diagrama representa como se recibe la información desde los diferentes módulos del SCADA al HMI, todas las implementaciones concretas del middleware son las responsables de recibir dichos flujos de datos, y la clase IOManager de proveérselo a los objetos interesados mediante diferentes formas en dependencia del tipo de dato, para la actualización del valor de las variables de la base de datos de tiempo real en los despliegues se tiene el concepto Updateable que lo implementan todos lo Widgets que

van a mostrar información en tiempo real, esto se hace mediante el objeto InputSlot que tiene una referencia a todos los objetos que implementan Updateable. Mediante la clase IOManager los widgets pueden suscribirse o eliminar dicha suscripción.

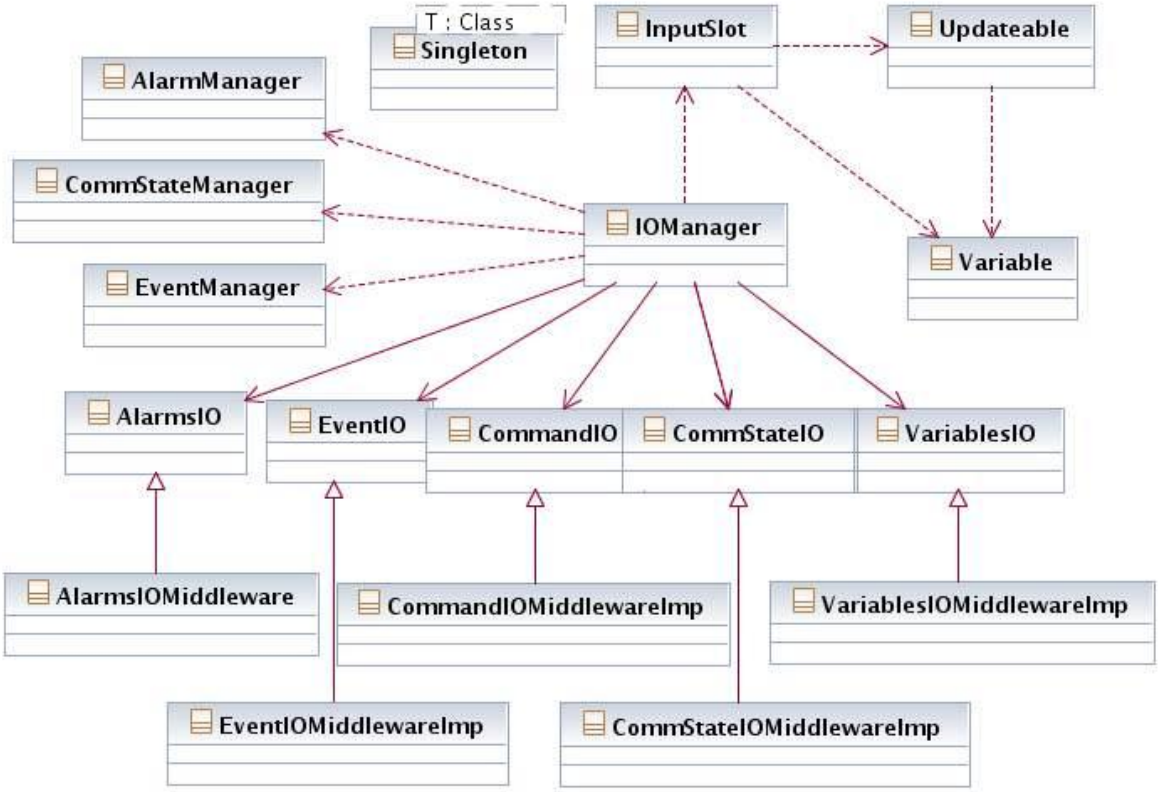

Fig. 14 Diagrama general del mecanismo Entrada/Salida.

Descripción diagrama "Mecanismo adquisición variables HMI Ejecución".

Diagrama que muestra el mecanismo de adquisición de variables del Módulo HMI Ejecución, se observa la clase IOManager responsable del mecanismo de Entrada/Salida (IO) la cual tiene una instancia de la clase VariablesIO que es la interfaz que debe implementar de acuerdo al mecanismo de adquisición que se utilice, en este caso se tienen los ejemplos de VariableIOMiddlewareImp diseñada específicamente para utilizar el middleware del SCADA y otra es la VariableIODummy que se utiliza como prueba en la simulación de datos de las cuales el sistema utilizará la que se instancie en la inicialización del sistema, la clase IOManager tiene también una colección de InputSlot que tiene una referencia a la variable que representa y a todos los objetos interesados en

la actualización los cuales deben implementar la interfaz Updateable que cada determinado tiempo actualiza dichos datos.

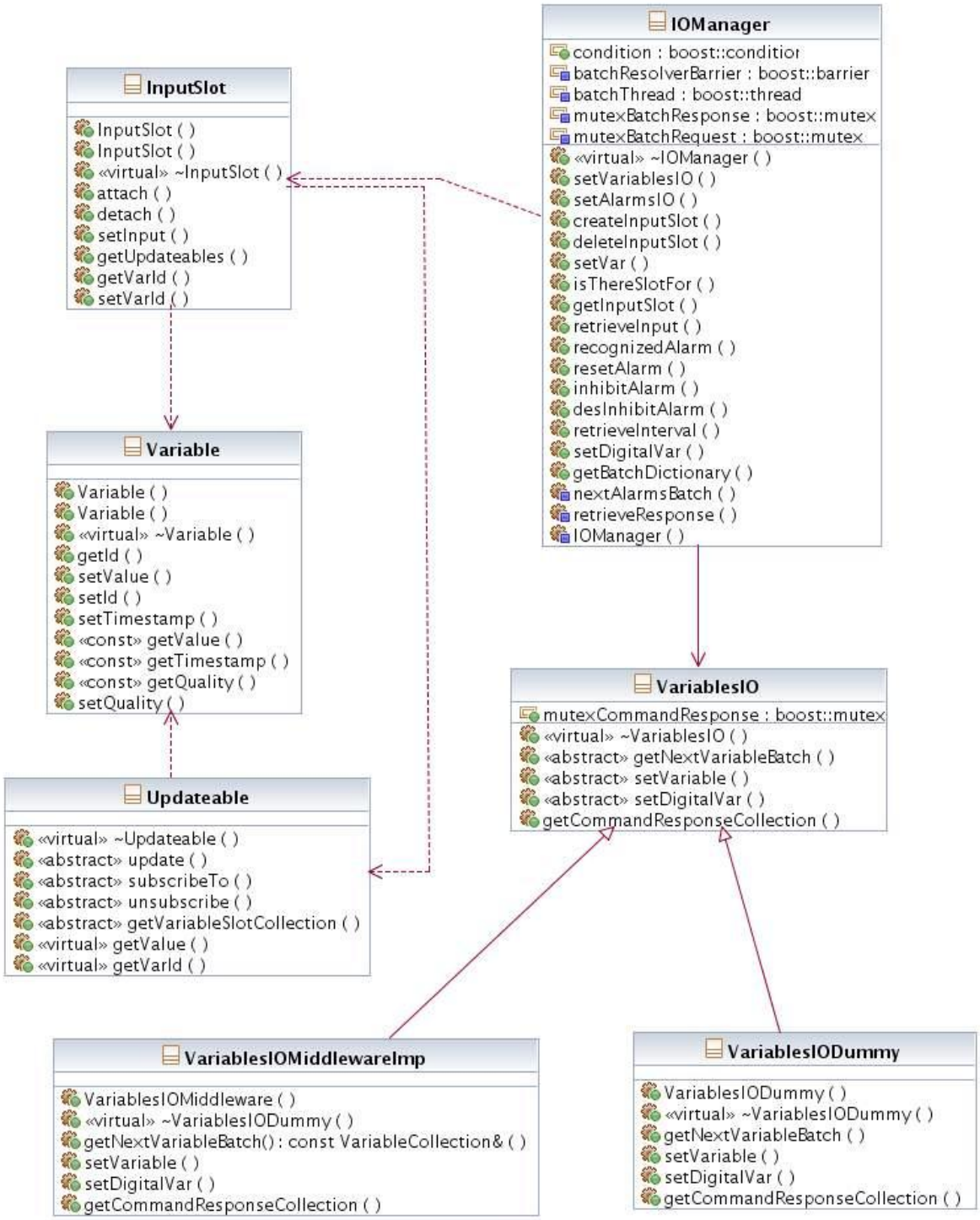

Fig. 15 Mecanismo Adquisición de variables HMI Ejecución.

Descripción diagrama "Mecanismo adquisición alarmas HMI Ejecución".

Diagrama que muestra el mecanismo de adquisición de alarmas del Módulo HMI Ejecución, la clase IOManager es responsable del mecanismo de Entrada/Salida (IO) la cual tiene una instancia de AlarmsIO que es la interfaz que debe implementar de acuerdo al mecanismo de adquisición que se utilice, en este caso el ejemplo de AlarmsIOMiddlewareImp diseñada para utilizar el middleware SCADA y la clase AlarmsIODummy con fines de simulación de datos para la realización de pruebas, de estas clases el sistema utiliza la que se instancie que puede ser una por vez. La clase IOManager también es responsable de los lotes de alarmas que se recolecten almacenarlos en la clase AlarmManager, que es utilizada como un repositorio de alarmas donde posteriormente los objetos interesados en procesarlas o mostrarlas tienen acceso.

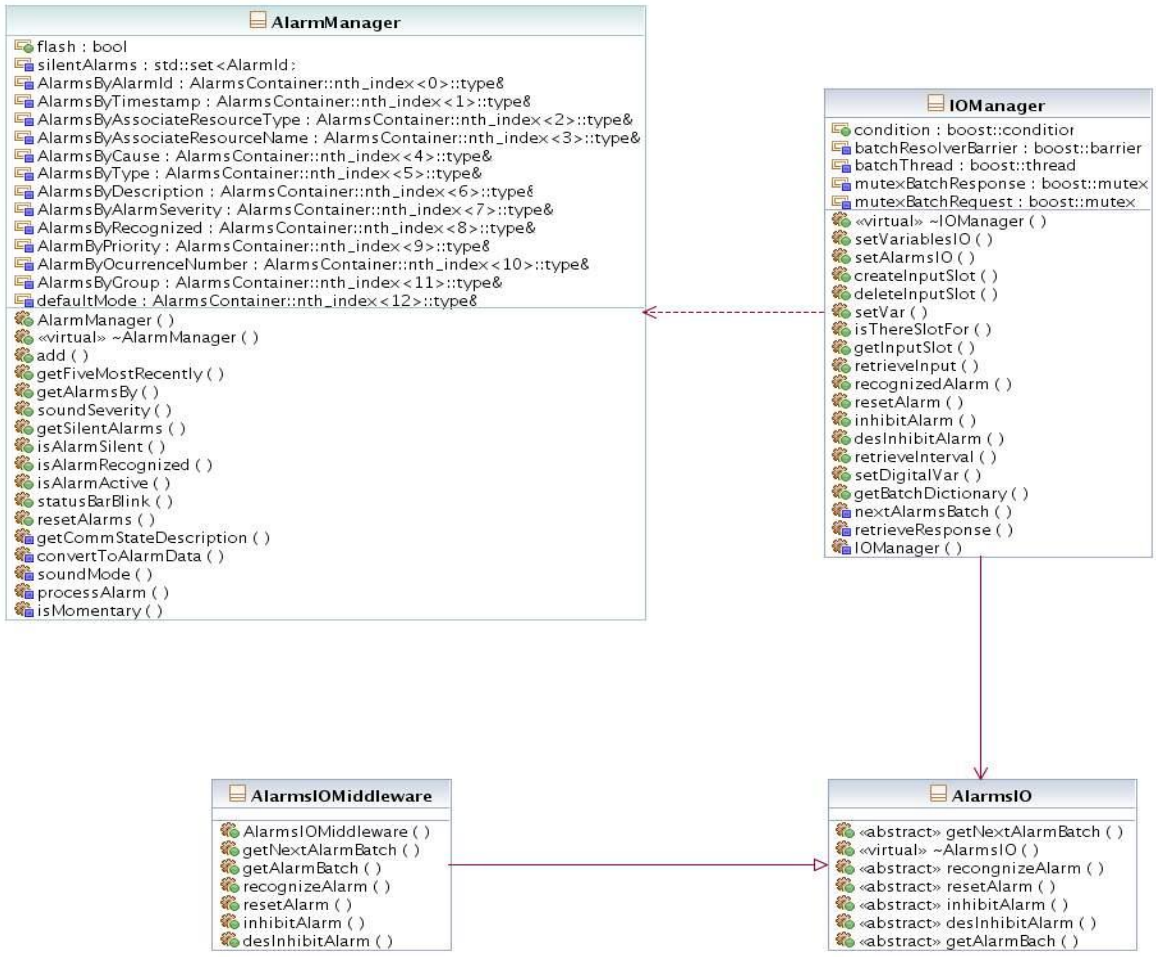

Fig. 16 Mecanismo de adquisición de alarmas HMI Ejecución.

Descripción diagrama "Mecanismo de procesamiento de comandos HMI Ejecución". Diagrama que muestra el mecanismo de envío de comandos y la recepción de la respuesta de estos. La clase IOManager es la responsable del mecanismo de Entrada/Salida la cual tiene una instancia de CommandIO que es la interfaz que se implementa de acuerdo al mecanismo de envío de comandos que se adopte en el sistema, en este caso se tiene CommandIOMiddlewareImp que es específicamente para el middleware usado en el SCADA.

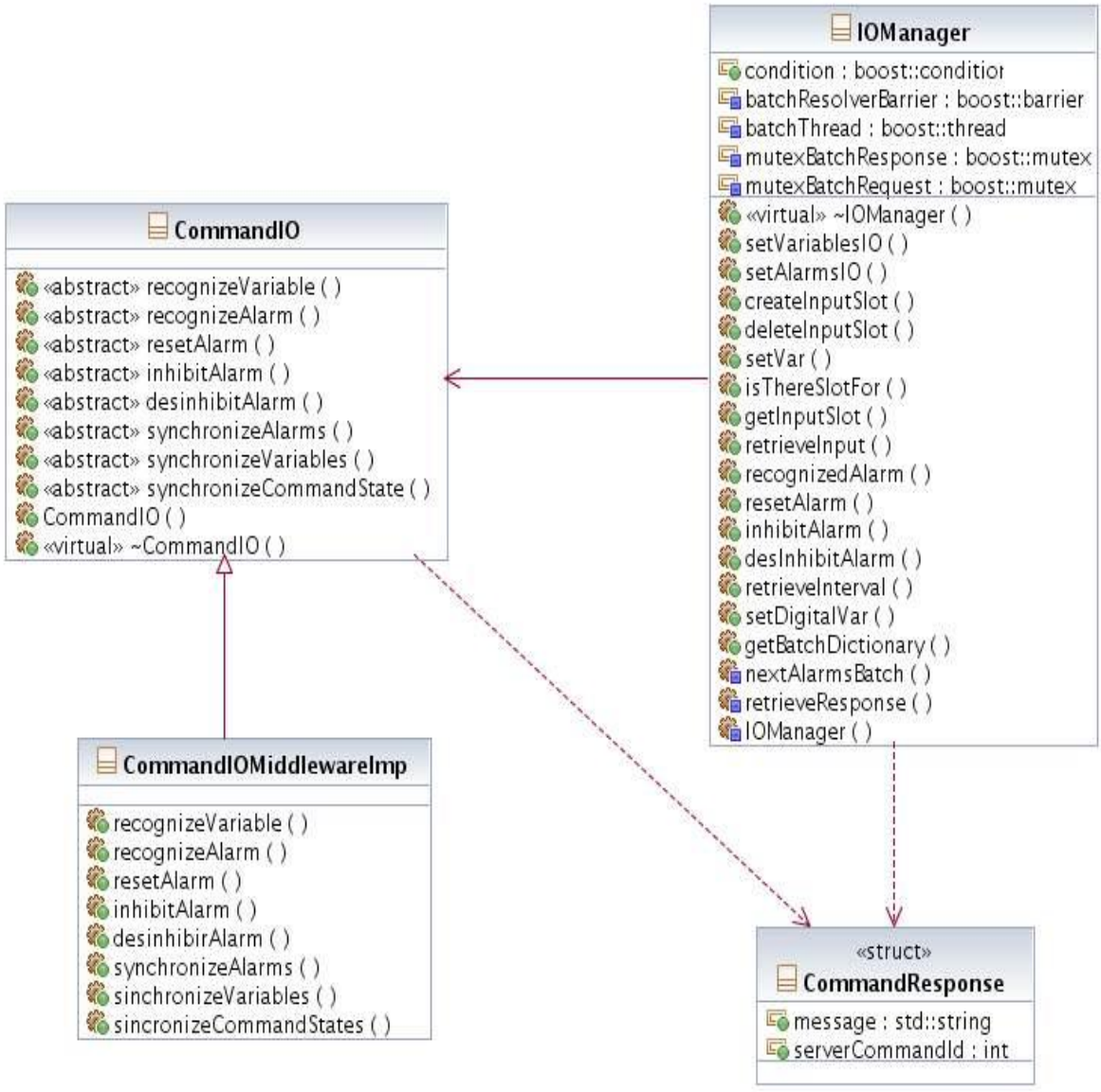

Fig. 17 Mecanismo de procesamiento de comandos HMI Ejecución

Descripción diagrama "Mecanismo de adquisición de eventos HMI Ejecución".

La clase IOManager tiene una instancia de la clase EventIO que es la que permite acceder a los eventos del sistema, la clase EventIOMiddlewareImp que es específica para el middleware del SCADA y la forma en que obtiene los eventos es sincrónica por lo que los objetos interesados en conocer los eventos del sistema solo tienen que pedirlo a la clase IOManager.

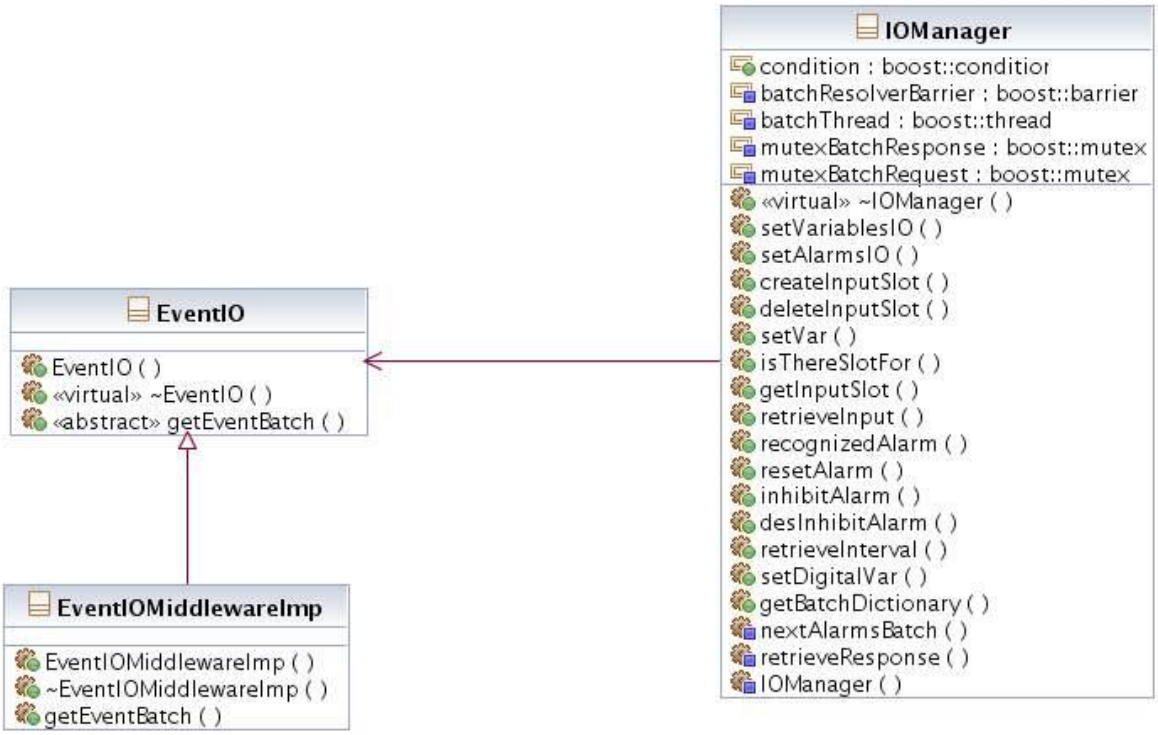

Figura 29 Mecanismo de adquisición de eventos HMI Ejecución.

Descripción diagrama "Mecanismo de adquisición de estado de comunicación HMI Ejecución".

La clase IOManager tiene una instancia de CommSatateIO que es la que define la interfaz que deben cumplir las implementaciones según el mecanismo que se utilice para la adquisición de datos, en este caso se encuentra diseñada la instancia concreta para el middleware del SCADA que se construye.

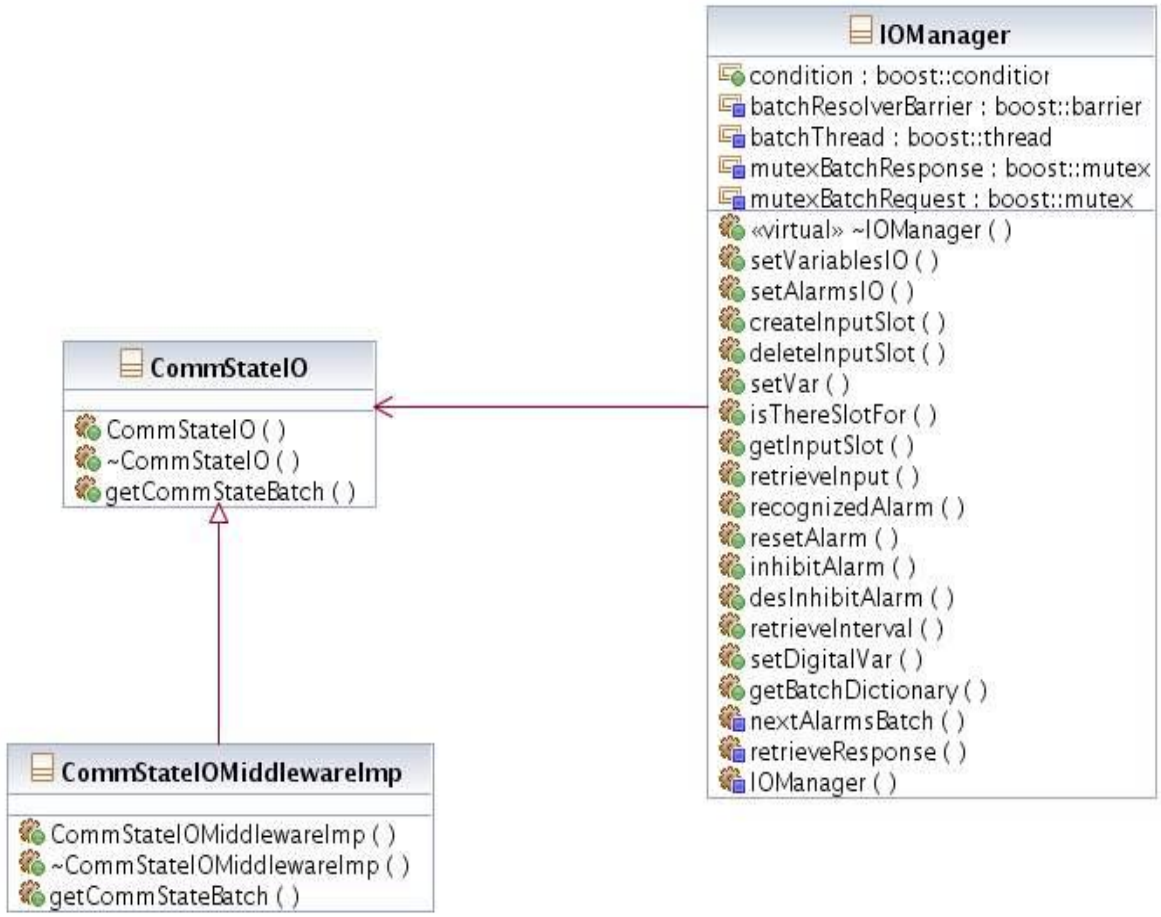

Fig. 18 Mecanismo de adquisición de estado de comunicación HMI Ejecución.

3.3.4 Descripción diagrama "Sumarios del SCADA HMI".

Diagrama donde se visualiza completamente el de forma general de los Sumarios del SCADA. En este diagrama se muestra la solución propuesta a los Sumarios de Alarmas, Eventos, Puntos, Estado y Estadísticas. Está diseñado para permitir adicionar funcionalidades o nuevos sumarios reutilizando las propiedades y características comunes que se pueda detectar.

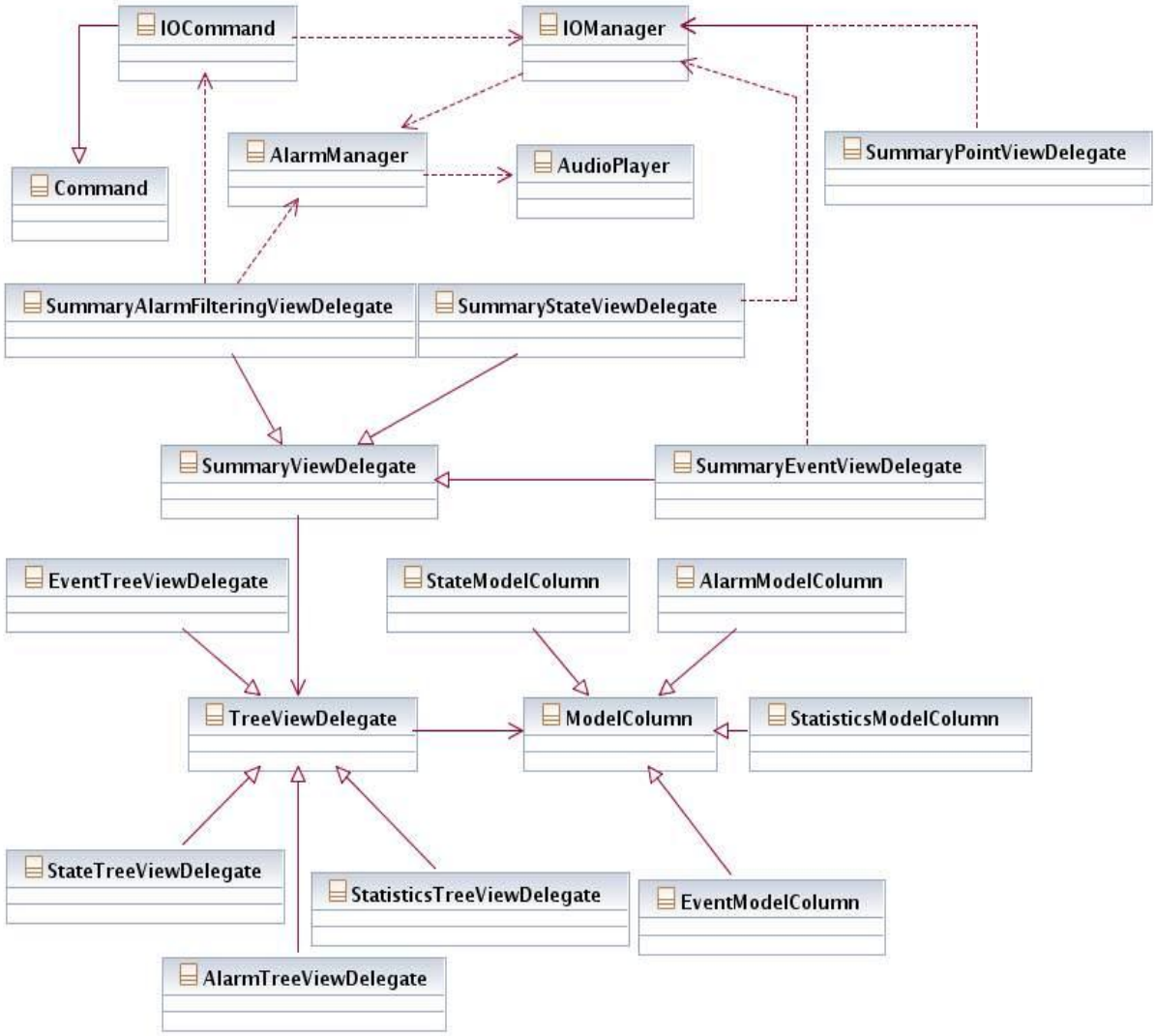

Fig. 19 Sumarios del SCADA.

Diagrama Sumario de Eventos.

Como su mismo nombre lo expresa este sumario es un despliegue de visualización de los eventos en el sistema. El mismo debe establecer una conexión con el modulo históricos y desplegar la información referente a los eventos almacenados en el Sistema en orden cronológico LIFO.

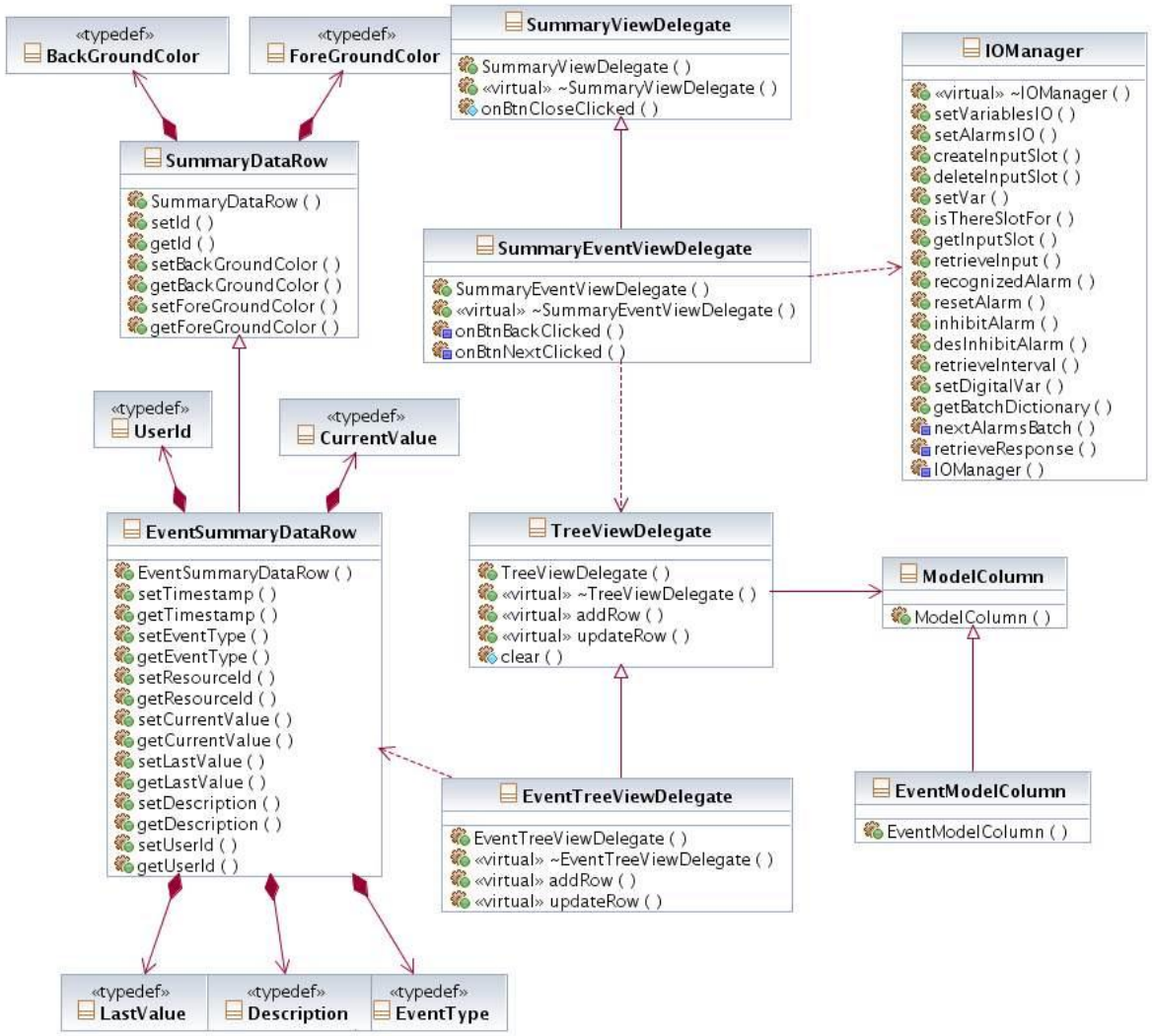

Fig. 20 Sumario de Eventos.

Diagrama Sumario de Estado de Dispositivos y Sub-canales.

Son sumarios que permiten el monitoreo del estado actual, en tiempo real de todos los dispositivos de control, configurados en el sistema, además de permitir ejercer acciones asociadas a los mismos, tales como habilitar/deshabilitar, etc. En este sumario se debe garantizar que exista un botón para visualizar el despliegue del sumario de estadísticas tanto para la de dispositivos como de sub-canales.

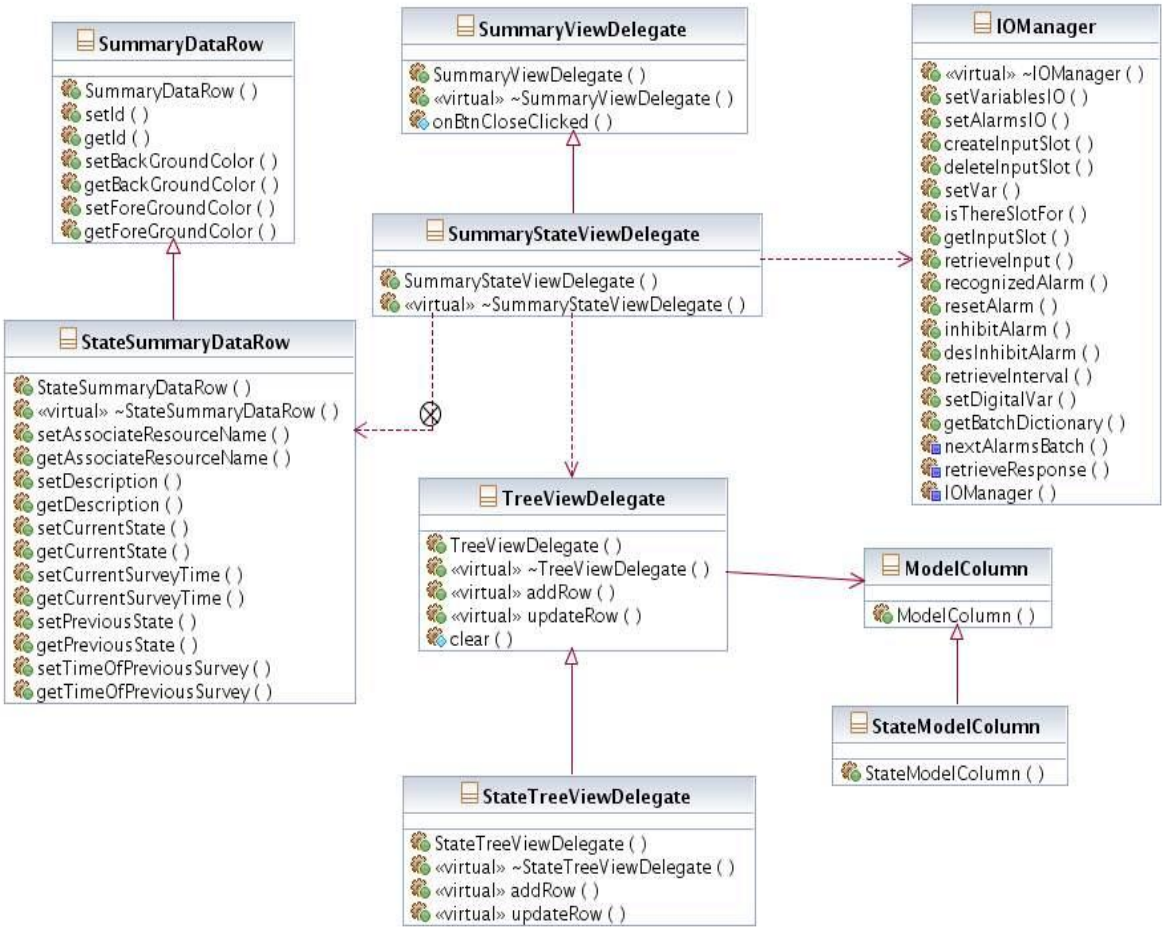

Fig. 21 Sumario de Estado.

Diagrama Sumario de Alarmas

El sumario de alarmas es una interfaz gráfica que concentra las condiciones de procesos críticas, medias y baja presentes en el sistema y cuyo objetivo primordial es guiar al operador a la detección del origen de la falla y supervisar la ejecución de las medidas de corrección automatizada o manual.

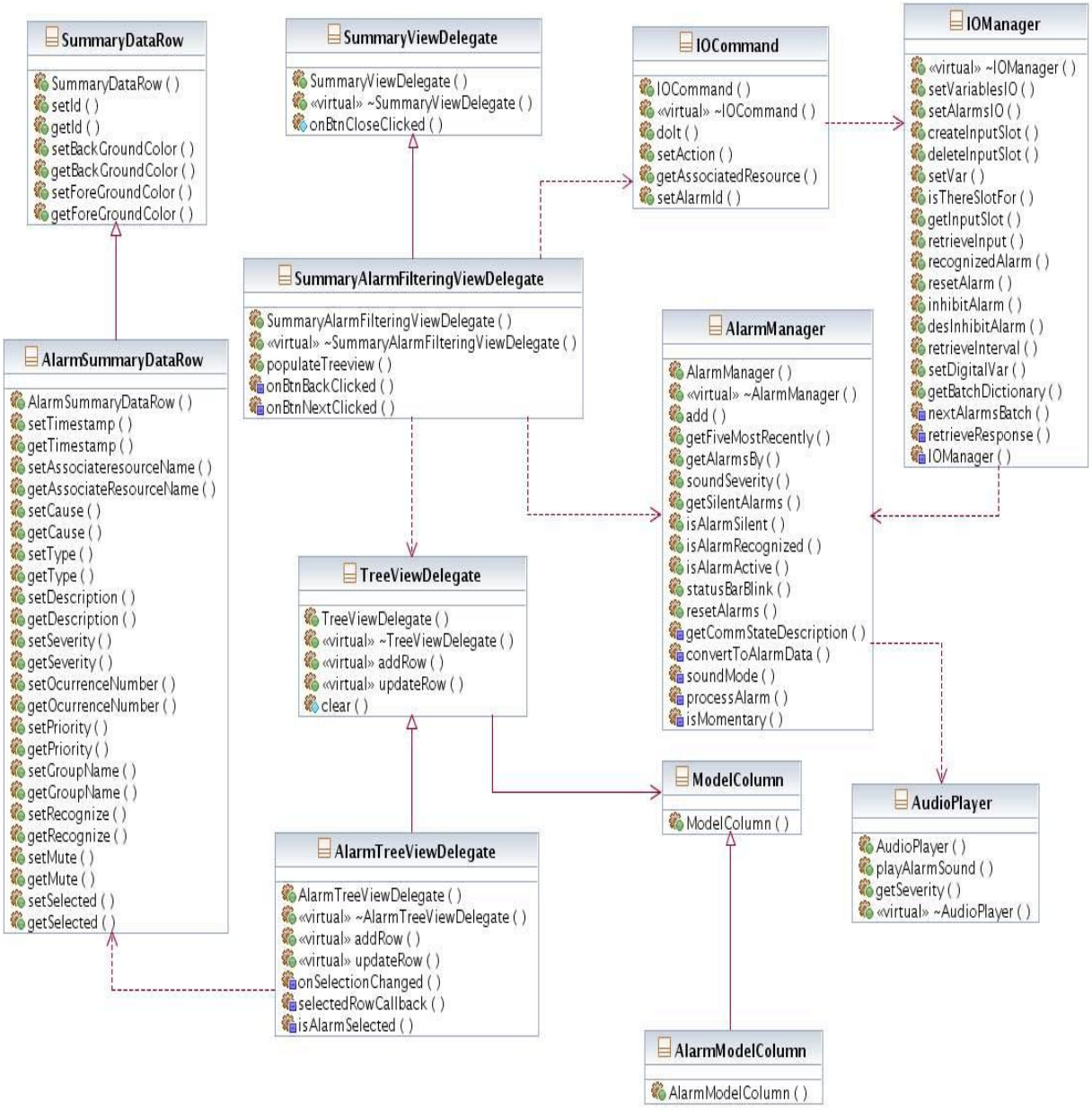

Fig. 22 Sumario de Alarmas.

Diagrama Sumario de estadísticas de comunicación de dispositivos y Sub-canales.
Son sumarios que nos permiten en base a los datos almacenados en históricos, conocer las estadísticas de comunicación de los dispositivos y sub-canales en el sistema, para ello muestran contadores de estados y porcentajes de éxito para las comunicaciones. En este sumario se debe garantizar que exista un botón para visualizar el despliegue del sumario de estado de los dispositivos y sub-canales.

Fig. 23 Sumario de Estadísticas.

Diagrama Filtros para los Sumarios del Sistema.

Todos los Sumarios del SCADA tienen como requisito un sistema que permite filtrar y ordenar la información mostrada al operador. Las especificaciones para estos filtros son:

Los filtros que podrán aplicarse a los sumarios de eventos, puntos, alarmas, estado y estadísticas, corresponden a los campos mostrados en las columnas de los parámetros generales para cada tipo de sumario. De igual forma, se debe garantizar la posibilidad de generar búsquedas con las combinaciones de éstos. Se utilizará la misma interfaz gráfica para mostrar los filtros aplicables y los resultados de las búsquedas realizadas de acuerdo a los grupos operaciones de privilegios asociados al perfil del usuario.

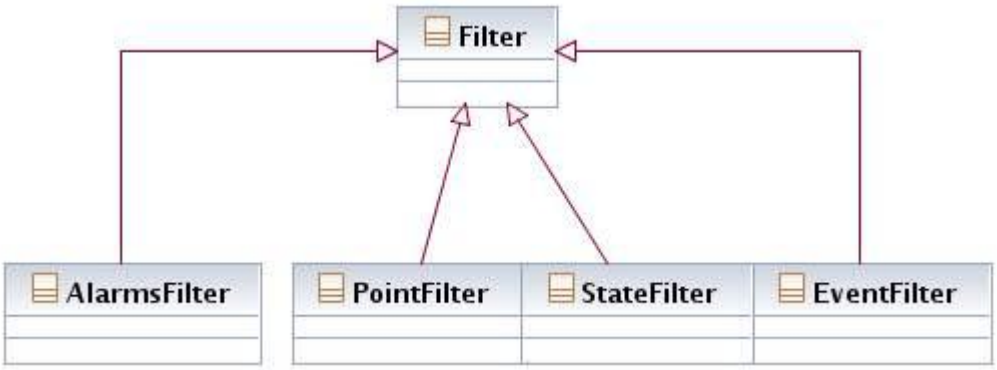

Fig. 24 Filtros Sumarios del SCADA.

Conclusiones

Con la realización del presente trabajo se arriban a las siguientes conclusiones.

- El estudio de las herramientas y tecnologías permitió realizar la ingeniería inversa y la modelación mediante UML del software.
- La definición de la estrategia de trabajo logró organizar y realizar toda la investigación que se requería para la realización del análisis y la obtención de la solución propuesta.
- Con el estudio de las aplicaciones con características similares a la que se desarrolla se obtuvieron diseños equivalentes a los deseados, los cuales sirvieron como punto de partida.
- De la obtención de la ingeniería inversa se obtuvieron los diagramas de clases que sirvieron para analizar el estado en que se encontraba la arquitectura.
- Con la documentación generada a partir de los distintos diagramas obtenidos y las descripciones que se realizaron se obtuvo la documentación de la arquitectura que sirve como guía a los integrantes del equipo de desarrollo.
- Mediante el análisis crítico que se realizó a la arquitectura del sistema se obtuvo un sumario de problemas a los cuales se trató de dar solución.
- Mediante la investigación de los patrones de diseño se comprendió su uso para luego aplicarlos a los modelos generados en la solución.
- La revisión de los requerimientos no funcionales sirvió para asegurarse que los diseños estaban optimizados para cumplir con las exigencias que estos plantean.
- Con la realización de los diagramas de diseño se obtuvieron las propuestas de mejoras que permitirían una biblioteca gráfica que pueda ser reutilizada en aplicaciones independientemente de si es de tipo SCADA, se propone un diagrama de componentes que hace el HMI más modular de forma que la compilación no se tenga que hacer sobre el sistema completo sino solo sobre los componentes modificados por el programador. Se obtiene el diseño de un subsistema de Entrada/Salida que cumple con los requerimientos de conexiones concurrentes y por distintos protocolos. Finalmente se obtiene el diseño completo de los Sumarios del SCADA que mediante la aplicación de los patrones de diseño estudiados permite adicionar nuevas funcionalidades y que estas reutilicen las características comunes que puedan presentar con los componentes que presenten relaciones.

Recomendaciones

Al concluir este trabajo se recomienda lo siguiente:

- Estudiar aplicaciones y tecnologías que permita obtener el conocimiento de cómo se desarrollan aplicaciones similares y aplicar las mejores prácticas que estas implementan.
- Aplicar los diseños obtenidos a la implementación del módulo Interfaz Hombre-Máquina del SCADA que se desarrolla para obtener las mejoras que estos proporcionan.
- Estudiar constantemente la evolución de la arquitectura del módulo con el objetivo de realizarle mejoras ya que es un proceso iterativo e incremental.

Referencias bibliográficas.

[1] Larman, C. UML Y PATRONES, Introducción al análisis y diseño orientado a objetos. México, Prentice Hall, 1999. 507pág.

[2] Booch, G., Rumbaugh, J., Jacobson, I. El Proceso Unificado de Desarrollo de Software. España, Addison Wesley, 2000. 458 pág.

[3] Pressman, Roger S. Ingeniería del Software. Un enfoque práctico. Madrid, Mac Graw Hill, 2001. 614 pág.

[4] Modelo vista Controlador. 2008. Disponible en: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

[5] JAOO, Århus, JavaZONE, Oslo. MVC XEROX PARC 1978-79. 2008. Disponible en: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

[6] Steve Burbeck, Ph.D, Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). 2008. Disponible en: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[7] GTK + TEAM, The GTK + Project. 2008. Disponible en: http://www.gtk.org/

[8] Valmet Automation. Manual de Operación OASyS Versión 6.0.6. Houston, Hollister, 1997. 314 pág.

[8] Rodriguez Penin, Antonio. Sistemas SCADA. Madrid, Marcombo, 2007. 447 pág.

[9] Romagosa Cabús, Jaume. Miniproyecto Automatización industrial. Cataluña, Universidad Politécnica de Cataluña. 2004. 66 pág.

[10] Progea. Movicon Programmer Guide. Modena – Italia. 632 pág.

[11] Figueras Solé, Enric. Memoria del proyecto de Ingeniería Técnica en informática de Sistemas. Barcelona. Universidad Autónoma de Barcelona. 60 pág.

[12] Fundación Eclipse, Proyecto Eclipse, www.eclipse.org/projects/project_summary.php?projectid=eclipse, 2008.

[13] IBM, Rational Software Architect, http://www-306.ibm.com/software/awdtools/architect/swarchitect/features/index.html?S_CMP=wspace , 2008.

[14] Interfaz Hombre-Máquina. 2008. Disponible en: http://www.es.wikipedia.org/wiki/SCADA

[15] Bergey, John;O´Brien, Liam & Smith, Dennis. Mining Existing Assets for software Product Lines. Pittsburgh. Software Engineering Institute, Carnegie Mellon University, 2000.

[16] Fowler, Martin. *Analysis Patterns: Reusable Object Models.* Adisson Wesley, 1997.

[17] Booch, Grady. OBJECT-ORIENTED ANALYSIS AND DESIGN. California, ADDISON-WESLEY, 1998. 543 pág.

[18] Christopher, Alexander. The Timeless Way of Building. Oxford University Press. 1979.

[19] Debian team, The Debian Project. 2008. Disponible en: http://www.debian.org/

Bibliografía

- Larman, C. UML Y PATRONES, Introducción al análisis y diseño orientado a objetos. México, Prentice Hall, 1999. 507pág.
- Booch, G., Rumbaugh, J., Jacobson, I. El Proceso Unificado de Desarrollo de Software. España, Addison Wesley, 2000. 458 pág.
- Pressman, Roger S. Ingeniería del Software. Un enfoque práctico. Madrid, Mac Graw Hill, 2001. 614 pág.
- Rodriguez Penin, Antonio. Sistemas SCADA. Madrid, Marcombo, 2007. 447 pág.
- Progea. Movicon Programmer Guide. Modena – Italia. 632 pág.
- Valmet Automation. Manual de Operación OASyS Versión 6.0.6. Houston, Hollister, 1997. 314 pág.
- Rodriguez Penin, Antonio. Sistemas SCADA. Madrid, Marcombo, 2007. 447 pág.
- Romagosa Cabús, Jaume. Miniproyecto Automatización industrial. Cataluña, Universidad Politécnica de Cataluña. 2004. 66 pág.
- Bergey, John;O´Brien, Liam & Smith, Dennis. Mining Existing Assets for software Product Lines. Pittsburgh. Software Engineering Institute, Carnegie Mellon University, 2000.
- Booch, Grady. OBJECT-ORIENTED ANALYSIS AND DESIGN. California, ADDISON-WESLEY, 1998. 543 pág.
- Christopher, Alexander. *The Timeless Way of Building.* Oxford University Press. 1979.
- Fowler, Martin. *Analysis Patterns: Reusable Object Models.* Adisson Wesley, 1997.
- Gamma, Erich, Elements of Reusable Object-Oriented software. Addison Wesley, 1994.

Glosario de términos

Adquisición de datos: Consiste en recolectar un conjunto de variables medidas en forma física a través de diferentes elementos (sensores, transductores...) para ser convertidas en digital y que puedan ser utilizadas por el computador.

Apache: Servidor HTTP de dominio público basado en el sistema operativo Linux. Se desarrolló en 1995 y es actualmente uno de los servidores HTTP más utilizados en la red.

Arquitectura del sistema: Describe cómo las funcionalidades del mismo se distribuyen sobre un número de componentes lógicos y cómo estos componentes se interrelacionan.

Arquitectura: La Arquitectura de una Empresa es el proceso mediante el cual se definen los aspectos a ser considerados para el uso de la información que apoya el funcionamiento del negocio, así como el plan para materializar tal proceso, su custodia y evolución. Las arquitecturas definen, no diseñan; de manera que los diseños de sistemas, bases de dato o redes que se llegaran a desarrollar, son consecuencia inmediata de las definiciones establecidas en las arquitecturas.

Autenticar: Verificación de la identidad de una persona o de un proceso para acceder a un recurso o poder realizar determinada actividad, así como también para detectar la veracidad del origen de un mensaje electrónico.

Base de Datos en Tiempo Real (BDTR): Estándar de actualización para los sistemas de supervisión y control en línea capaz de manejar típicamente información proveniente de estaciones de flujo, pozos, múltiples de producción, entre otros. El término tiempo real está asociado a la frecuencia con la cual se deben muestrear los datos y que debe ser lo suficientemente rápido para lograr caracterizar el proceso.

Base de datos histórica: Es una base de datos que almacena variables analógicas, enteras o de tipos especializados, y que se registran en términos del tiempo, fecha y hora. Los datos sólo son incluidos y nunca modificados, registrando así los eventos asociados al mismo en el tiempo.

Casos de Uso: Proporcionan uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico, utilizando un lenguaje más cercano al usuario final. También, se trata de una

secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema.

Capa: Es un grupo de componentes diseñados de tal manera que puedan ser reutilizables en circunstancias similares.

Clase: En programación orientada a objetos, es un tipo de datos definido por el usuario que especifica un conjunto de objetos que comparten las mismas características. Tienen unos atributos que lo caracterizan y métodos que describen la funcionalidad.

Compilador: Es aquel que acepta programas escritos en un lenguaje de alto nivel y los traduce a otro lenguaje, generando un programa equivalente independiente, que puede ejecutarse tantas veces como se quiera. Este proceso de traducción se conoce como compilación.

Dato: Término que generalmente es usado para describir las señales con las cuales trabaja el computador. Se trata de la unidad de información mínima no elaborada, sin sentido en sí misma, que convenientemente tratada se utiliza para realizar cálculos o toma de decisiones.

Diagramas de casos de uso: Sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/o otros sistemas.
Diseñar: Implica la ejecución de una serie de pasos que originan un producto conocido como objeto de diseño.

Estándar: Especificación que regula la realización de ciertos procesos o la fabricación de componentes para garantizar la interoperabilidad, también conocido como norma.

Evento: Conjunto de acciones que proceden de la ejecución o activación de otra acción y que ha sido registrada, la combinación de estas acciones también pueden dar como resultado otro evento en particular o una serie de eventos.

Integración de aplicaciones: Define la forma de interacción entre las aplicaciones externas del SCADA y algunos componentes internos. Puede ser realizado mediante un conjunto de funciones o una capa “middleware” que permita la comunicación entre diferentes componentes de la arquitectura.

Interface: Se trata del dispositivo o elemento intermedio (hardware o software) que permite la comunicación entre los sistemas.

Interfaz Gráfica de Usuario (GUI): Están conformados por despliegues que permiten una mejor y más fácil interacción con el computador. Además, contribuye en el aprendizaje intuitivo de los programas, facilitando y reduciendo el tiempo de formación y aumentando la productividad.

Interfaz Hombre Máquina (IHM): Consiste en herramientas para la supervisión y control del proceso (consolas de operación y generador de despliegue).

Lenguaje de Marcado de Hipertexto (HTML): Es un lenguaje de marcación diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas Web.

Lenguaje interpretado: Lenguajes que suelen ser ejecutados por un intérprete, en lugar de ser compilados. Permite independencia entre el programa y la plataforma en la cual se ejecuta. Ejemplo: php, perl, phyton, javascript, entre otros.

Licencia Pública General (GPL): Licencia Pública General desarrollada por la FSF (Free Software Foundation), permite instalar, modificar, usar y distribuir programas GPL en uno o varios equipos sin limitación.

Multiplataforma: Sistema que puede ejecutarse en diversos sistemas operativos y tipos de hardware.

Protocolo de transferencia de hipertexto (HTTP): Utilizado en Internet y conocido como WWW.

Requerimiento: En ingeniería de software, se define como "una condición o capacidad que un sistema debe cumplir".

Sistema distribuido: Es un sistema desarrollado de forma modular (por componentes) que permite el funcionamiento de los módulos en nodos diferentes, garantizando la comunicación entre ellos.

Time stamp: Tiempo en que ocurre o se ejecuta un evento.

Usuario: Persona que interactúa con el computador a nivel de aplicación.

Alarma: Evento generado por un dispositivo o una función que señala la existencia de una condición anormal a través de un cambio discreto audible o visible, o ambas, que requiere su atención inmediata.

Comandos: Acciones que se llevarán a cabo en tiempo de ejecución del SCADA y que influirán sobre los recursos del sistema (despliegues, variables, alarmas, usuarios). Los comandos definen funcionalidades, utilizadas comúnmente en la creación de aplicaciones de supervisión sin la necesidad de programación de scripts, ejemplo de comandos son: abrir un despliegue, mostrar un menú, asignar un valor a una variable etc. Los comandos puedes ser clasificados según su funcionalidad como: comandos de variable (cambiar un variable), comandos de sistema (ejecutar un aplicación externa), comandos de usuario (abrir sesión), comandos de despliegues (mostrar un despliegue), comandos de alarmas (reiniciar, reconocer), comandos de reporte (mostrar un reporte).

Control: Se refiere al conjunto de acciones que permiten modificar el estado actual de los actuadores en campo a través del envío de comandos.

Controlador Lógico Programable (PLC): Dispositivos electrónicos usados en automatización industrial para realizar estrategias de control básicas. Por su robustez y características sencillas de control, están cercanas al proceso, permitiendo ejecutar las tareas básicas del control, aun cuando no tenga conexión a las capas superiores del control.

Despliegue (operacional): Provee una navegación sencilla de acuerdo a una secuencia lógica del flujo del proceso. Parte integral de la interfaz gráfica de usuario.

Punto: Los puntos se refieren a las direcciones de memoria que se configuran en el SCADA y que referencian a una variable capturada por un dispositivo de campo, o una variable del proceso. Pueden ser de tipo analógico, digital, texto y vector.

Sub-canal: Es parte constitutiva de un canal de transmisión, en el SCADA, es utilizado para agrupar lógicamente los dispositivos de campo que utilizan el mismo protocolo y que por consiguiente son accedidos por un mismo manejador.

Tagname: Es una etiqueta asignada a una variable a ser medida. Por regla general, la estructura de la etiqueta se compone de elementos que permiten describirla: tipo de variable, ubicación geográfica y lógica, entre otros.

Telemetría: También conocida como "medición a distancia", consiste en medir ciertos parámetros (temperatura, presión, nivel...) utilizando elementos (sensores) que envían los datos de manera remota a un receptor para su análisis.

Unidad de campo: Es utilizada para mostrar la señal proveniente de campo en formato binario, de acuerdo a los rangos establecidos en el dispositivo de control.

Unidad de ingeniería: Es utilizada para mostrar las indicaciones de los instrumentos de medida de acuerdo a los sistemas de medición definidos para el sistema (por ejemplo, sistema inglés de unidades o sistema internacional de medidas).

Unidad Terminal Remota (RTU): Dispositivos basados en microprocesadores, el cual permite obtener señales independientes de los procesos y enviar la información a un sitio remoto. Las RTU en los tiempos han sido desplazadas por los PLC quienes han fortalecido sus facilidades de comunicación a través de protocolos para sistemas de control (MODBUS, DNP3, IEC-101, etc.)

Widget (Objeto gráfico): Objeto gráfico que puede contener gráficos vectoriales o imágenes raster, (imágenes de mapas de bits) y presenta un conjunto de propiedades que pueden ser editadas y asociadas a puntos (variables) del SCADA o a expresiones que contengan variables del sistema. Por medio de los objetos gráficos se pueden crear animaciones en función de los valores de los puntos asociados. Los widgets pueden agruparse para formar nuevos objetos gráficos más complejos. Un ejemplo de widget puede ser un contenedor de textos.

Cuadro Sinóptico: Exposición de una materia en una plana, en forma de epígrafes comprendidos dentro de llaves u otros signos gráficos, de modo que el conjunto se puede abarcar de una vez con la vista.