Facultad 5

Diseño e implementación del modelo CIM en el HMI del SCADA SAINUX

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

Autor: José Manuel Castellanos Mecías

Tutores: Ing. Henry Marcelo Cabrera Robles

MSc. Manuel Villanueva Betancourt

Co-Tutor: Ing. Yoandri Matos de la Cruz

La Habana, Julio de 2016

DECLARACIÓN DE AUTORÍA

Por este medio declaro ser autor de este trabajo y reconozco a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Firma del Autor

José Manuel Castellanos Mecías

_____ _____

Ing. Henry Marcelo Cabrera Robles Msc. Manuel Villanueva Betancourt

Firma del Tutor Firma del Tutor

_____

Firma del Co-Tutor

Ing. Yoandri Matos de la Cruz

AGRADECIMIENTOS

Agradezco a todas las personas que junto a mi lloraron y lucharon en los momentos más difíciles para que pudiera salir adelante y convertirme en la persona que soy hoy.

Agradezco también a los que sin darse cuenta me alegraron el día con una sonrisa y a los que supieron tener la paciencia de un padre o una madre para tolerar mis errores y mis momentos de insensatez frente a la vida. A los que me dieron la mano o incluso de comer cuando más lo necesitaba y los que pusieron su confianza en un joven inexperto con tan pocos sortilegios como para develar esperanzas.

Muchas gracias a todos aquellos que contribuyeron durante estos cinco años a mi formación profesional y personal, pero sobre todo a los que tuvieron la estirpe de héroe para impregnar sabiduría en quién dudó en más de una ocasión durante estas 5 batallas, porque ellos como Mella, fueron capaces de hacer tanto en tan poco tiempo.

A mi familia

No alcanzaría el vocabulario de la lengua española para expresar mi agradecimiento por tanto sacrificio, amor y comprensión durante 24 años. 24 años que supieron avivar en mí el fervor de un hijo, hermano e incluso padre, porque para ellos soy la razón de existir y para mi ellos son el adalid a seguir cuando mis días carecen de sentido. Los amo, y su decepción sería para mí, el único error que jamás perdonaría un mortal como yo, porque mi orgullo es sentirme parte de esa maravillosa familia.

A mis Amigos

Quisiera decir que son mis amigos, pero no me salen tales palabras, quisiera decir que fueron mis compañeros, pero estaría mintiendo. Ustedes han sido y serán siempre mi familia; hermanos con los que reí, lloré y perdí el sentido bajo los efectos de algún producto que en ocasiones era indistinguible. Y por supuesto no hablo solo de esos amigos que durante 5 años sufrieron de mis manías sino de todos a los que alguna vez llamé hermanos, pues de esos existen pocos y tengo el placer de sentir que cuento con ellos incondicional y eternamente al igual que será mi amistad con ustedes.

DEDICATORIA

Para las personas que arrancaron el motor de la superación personal y para quienes supieron mantenerlo vivo durante todos mis estudios.

***A mi abuelo** que tuvo la visión de un hombre de ciencia en un niño cuando apenas tenía 10 años.*

***A mi tío** que me dio herramientas para no padecer burlas. A quien me enseñó el "idioma habanero"*

***A mis profesores** que lograron sembrar la semilla de la educación en tan yermo terreno.*

***A mis amigos** Oscar, Diome, Jorgito, Luis, Laura, David, Dayron, Denys, Javier, José Carlos, Daylen, Yanecita, y todos los demás compinches que son hoy, más que amigos para mí. Ustedes supieron darme el apoyo y el amor que necesitaba cuando estaba lejos de mi hogar, ustedes supieron en su momento, darme un hogar.*

***A mi hermana** Heidy, **a mis abuelos** Daysi y José, **a mis tíos** porque todos de una forma u otra aportaron a que yo me convirtiera en el hombre que soy.*

***A mi padre** que me dio no solo amor, sino la fuerza que me hace levantarme en la vida y encarar los problemas con el mayor ímpetu sin temor a vacilar, por más difícil que sea el obstáculo.*

***A la niña de mis ojos** que es y siempre será la mejor amiga de todas, la mejor madre, la mejor compañera y la persona que adoro desde que abrí los ojos por primera vez y seguiré haciéndolo hasta el día que los cierre. A ti te debo cada latido de mi corazón y cada sorbo de aire de mi vida y si el mundo se voltea y todo me queda de cabeza la única persona que seguirá recta, como el faro que guía mi vida, serás tú mamá.*

Gracias por todo

RESUMEN

El auge de la informatización de los procesos cobra, cada día, mayor fuerza en las industrias, fundamentalmente en nuestro país. Basado en la mejora de la calidad y la eficiencia en la producción, se desarrollan productos informáticos capaces de garantizar el cumplimiento eficaz de los planes económicos y políticos del gobierno cubano, reflejados en los lineamientos del Sexto Congreso del Partido Comunista de Cuba que convergen a la integración de nuestras necesidades con el crecimiento mundial en esta rama.

La Universidad de las Ciencias Informáticas cuenta con el Centro de Informatización Industrial, que desarrolla soluciones informáticas para satisfacer las emergentes demandas del mercado y entre sus productos en desarrollo está el SCADA SAINUX. El presente trabajo de investigación tiene como objetivo el manejo coherente de la información en los módulos pertenecientes a la Interfaz Hombre-Máquina, posibilitando de esta forma la corrección de deficiencias como la información redundante, el alto acoplamiento de las clases y el manejo de las asociaciones, logrando mejorar las operaciones de actualización en los despliegues y la correcta liberación de la memoria.

En la solución es usado el lenguaje de programación C++, el marco de trabajo (*framework*) Qt, como lenguaje de modelado UML y siguiendo la metodología AUP-UCI. Se muestra además las especificidades del SCADA SAINUX y su modelo de datos, se identifican los principales patrones de diseño y arquitectura, proponiendo buenas prácticas en la implementación. Con la aplicación de pruebas de aceptación, rendimiento y mantenibilidad se pudo comprobar que la investigación cumple con las necesidades existentes y su futura integración permitirá un mejor manejo y gestión de la información dentro de los módulos del HMI.

PALABRAS CLAVE

Interfaz Hombre-Máquina, modelo de datos, marco de trabajo, módulo, SCADA.

ABSTRACT

The rise of computerization of the processes gains, every day, more strongly in industries, mainly in our country. Based on improving quality and production efficiency, computer products capable of ensuring the effective implementation of economic and political plans of the Cuban government, reflected in the guidelines of the Sixth Congress of the Communist Party of Cuba converging develop the integration of our needs with global growth in this branch.

The University of Information Science has Informatization Industrial Center (CEDIN), which develops solutions to meet emerging market demands and among its products in development is the SCADA SAINUX. This research aims coherent information management in the modules belonging to the Man-Machine Interface (HMI), thus enabling correction of deficiencies such as redundant information, the high coupling of classes and management associations, managing to improve operations update deployments and proper release of memory.

In the solution is used the programming language C ++, the framework Qt as UML modeling language and following the AUP-UCI methodology. SAINUX the specificities of SCADA and data model also shows, major design and architecture patterns are identified, proposing good practices in implementation. With the application of acceptance testing, performance and maintainability it was found that the research meets existing needs and future integration will enable better management and information management within these modules HMI.

KEYWORDS

data model, framework, Man-machine interface, module, SCADA.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La informatización de la sociedad cubana es una voluntad política del gobierno que está expresada en los Lineamientos de la Política Económica y Social, aprobados en el Sexto Congreso del Partido Comunista de Cuba. En varios sectores de la industria y la economía cubana se necesita un sistema capaz de informatizar la supervisión y control de sus procesos. SAINUX[1], es un sistema SCADA distribuido, orientado a componentes y en proceso de desarrollo por especialistas del Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas (UCI).

Un sistema de supervisión, control y adquisición de datos SCADA (acrónimo de *Supervisory Control And Data Acquisition, por sus siglas en inglés*) se refiere a un sistema central que monitorea y controla un sitio completo. Especialmente diseñado para funcionar sobre ordenadores en el control de la producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, entre otros). La instalación de un sistema SCADA necesita un *hardware* de señal de entrada y salida, sensores y actuadores, controladores, HMI[2], redes, comunicaciones, base de datos, entre otros.

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

El HMI consta de dos entornos: el entorno de configuración (EC), donde los mantenedores configuran la información de la planta en cuestión en forma de proyecto. En este se configura un módulo de seguridad para controlar quien accede a que recurso y en qué momento. Además, se configuran tantos módulos de adquisición como se necesite para obtener los datos de campo que va a manejar el sistema y un módulo de aplicaciones para automatizar tareas. Finalmente, un módulo de HMI que es donde se maneja la información específica del área que se desea supervisar y se diseñan los despliegues, los cuales haciendo uso de los componentes gráficos permiten simular los procesos de campo; además está el entorno de visualización (EV) que es donde el operador puede supervisar y controlar la configuración realizada en el EC, interactuando con los componentes gráficos para emitir control sobre el sistema,

[1] Sistema de Automatización Industrial basado en GNU/LINUX
[2] Interfaz Hombre-Máquina

monitorear los cambios de estado de las variables, gestionar alarmas y generar reportes.

Para manejar toda esta información se utiliza el modelo CIM (del inglés, *Common Information Model*), ya que en éste se organiza la información que se va a necesitar en todo el sistema de manera lógica y estructurada. Actualmente en el SCADA SAINUX existe este modelo, pero tiene varios inconvenientes como son:

- Existe información redundante.
- Existe un alto acoplamiento entre las clases.
- Las relaciones de asociación no se manejan correctamente.

Todos estos problemas traen como consecuencia la dificultad de realizar operaciones de actualización en los despliegues e imposibilita la liberación correcta de la memoria. Por todo lo anteriormente planteado surge el siguiente **problema de investigación:** ¿Cómo contribuir a la organización y correcto manejo de información de los módulos del HMI del SCADA SAINUX?

Se presenta como **objeto de estudio:** el modelo de datos de un SCADA, que persigue como **objetivo general:** desarrollar una solución para el SCADA SAINUX que permita organizar la información del HMI mediante la implementación del modelo CIM. Enmarcado en el **campo de acción:** modelo de datos del HMI del SCADA SAINUX.

Tareas de Investigación:

- Elaboración del marco teórico de la investigación a través de la actualización del estado del arte que existe actualmente sobre el tema.
- Identificación y caracterización de toda la información que puede ser necesaria en la gestión de los sistemas SCADA.
- Realización del levantamiento de requisitos funcionales y no funcionales.
- Diseño e implementación del modelo de datos para el HMI del SCADA SAINUX.
- Realización de pruebas para validar el cumplimiento de los requerimientos.
- Elaboración del informe que contemple el proceso de investigación y desarrollo definido.

Posibles Resultados:

- Actualización del estado del arte del estándar CIM.
- Especificación de los requerimientos del Sistema.
- Diseño e implementación del modelo CIM para la organización de la información en el HMI del SCADA SAINUX.

Métodos científicos:

- **Métodos teóricos**
  - **Análisis histórico-lógico:** se utiliza para analizar la evolución histórica de soluciones similares en los Sistemas de Gestión Industrial, así como las técnicas utilizadas a nivel mundial, las tendencias actuales y su utilización en la UCI para llegar a las características necesarias y deseables para la solución que se propone.
  - **Analítico-sintético:** para el análisis de la documentación existente relacionada con el tema, extrayendo los elementos más importantes y necesarios para dar solución al problema existente, de manera que permita sintetizar todo lo obtenido relacionado con los estándares pertinentes y trabajos ya realizados anteriormente por expertos en el tema.
- **Métodos empíricos**
  - **Modelación:** se emplea para representar gráficamente la solución que se propone.
  - **Generalización:** permite sistematizar en cada capítulo de la investigación las pautas más significativas de la misma, así como llegar a conclusiones más objetivas y explícitas para cada caso.

Estructura capitular:

Capítulo 1: “Marco teórico de la investigación”; se detallan un conjunto de elementos para conformar el marco teórico relacionado con los aspectos definidos en el objeto de estudio. Se describe los principales conceptos de la investigación, precisando cuáles caracterizan un sistema SCADA y un estándar CIM. Además, se identifican y analizan las tecnologías y herramientas existentes, detectando sus mejores prácticas para tenerlas en cuenta en la solución a proponer.

Capítulo 2: “Análisis y diseño de la solución”, se exponen los requisitos funcionales y no funcionales, así como las características actuales del sistema para realizar una propuesta de solución a los problemas detectados. Se presenta la descripción de la

solución propuesta al problema de investigación, así como los diagramas y modelos elaborados para este producto.

Capítulo 3: “Pruebas y validación de la solución”; se realiza una descripción de la última fase de la metodología aplicada referente al correcto funcionamiento del producto, y se realizan las pruebas de caja blanca y rendimiento para validar los requerimientos del cliente.

CAPÍTULO 1: "MARCO TEÓRICO INVESTIGATIVO"

1. Introducción

En este capítulo se detallan un conjunto de elementos para conformar el marco teórico relacionado con los aspectos definidos en el objeto de estudio. Se describen los principales conceptos de la investigación, precisando cuáles caracterizan un sistema SCADA y un estándar CIM, así como las especificaciones de los módulos HMI. Además, se identifican y analizan las tecnologías y herramientas existentes, detectando sus mejores prácticas para tenerlas en cuenta en la solución a proponer.

1.1. Sistemas SCADA

A decir de Javier García Giménez *[*1*]*, "*los sistemas SCADA, utilizan el computador y las tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales. Estos sistemas son partes integrales de la mayoría de los ambientes industriales complejos o geográficamente dispersos ya que pueden recoger la información de una gran cantidad de fuentes rápidamente, y la presentan a un operador en una forma amigable*".

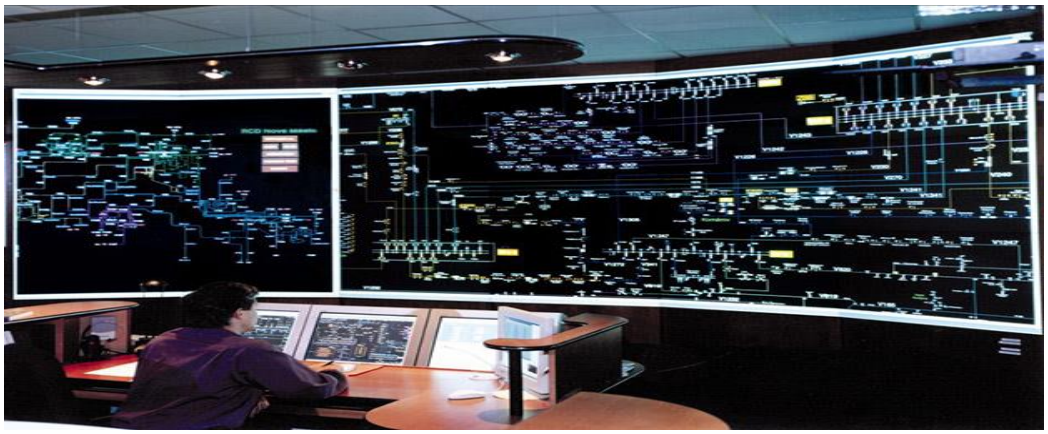


Figura 1: Operador interactuando con la Interfaz Hombre-Máquina de un sistema SCADA.

Se define entonces que un sistema SCADA es la integración de un conjunto de recursos tecnológicos que permiten la automatización y control de procesos complejos facilitando la obtención y análisis de datos recibidos por dispositivos de adquisición en áreas extensas.

En la actualidad los sistemas SCADA desempeñan un papel fundamental en la automatización industrial, pues con ellos se logra capturar información de un

proceso o planta industrial. Esta información es empleada para realizar análisis con los que se pueden obtener importantes indicadores que permiten una realimentación del proceso, esto repercute en la reducción de costes de mantenimiento de forma muy positiva, no así las funciones de control del servidor que están casi siempre restringidas a reajustes básicos del sitio o capacidades de nivel de supervisión.

De forma general se consideran como funciones básicas de un sistema SCADA la supervisión y control remoto de instalaciones, procesamiento de información, generación de reportes, gestión de alarmas, almacenamiento de información histórica, presentación de gráficos de tendencias y programación de eventos, entre otros.

Inicialmente las soluciones SCADA eran sistemas centralizados, es decir, todas las operaciones se realizaban en un solo ordenador reduciendo considerablemente la capacidad de respuesta y aumentando la carga de los dispositivos que soportaban la adquisición de datos. Tiempo después surge, con la incorporación de los ordenadores, la posibilidad de distribuir esta carga en otros dispositivos, manteniendo aún un control general en un servidor centralizado.

Un sistema distribuido no es más que un conjunto de elementos, tanto de procesamiento como de almacenamiento que están conectados a través de una red y que para un usuario del sistema se comporta como un único computador. Existen cinco características que son responsables de la utilidad de los sistemas distribuidos: compartir recursos, apertura, concurrencia, tolerancia a fallas y transparencia *[*5*]*.

Cuando hablamos de un sistema SCADA, no debemos centrarnos solo en las diferentes pantallas que nos muestran toda la información de forma sencilla e intuitiva. Detrás de todo esto está toda una amplia gama de elementos de supervisión y control, sistemas de comunicaciones y múltiples utilidades de *software* para hacer que el sistema funcione de forma segura, rápida y eficiente. A continuación, se relacionan un conjunto de ventajas que proporcionan los sistemas SCADA:

- El actual desarrollo de estos permite que se puedan utilizar sin ser experto en la materia.

- Los autómatas son altamente configurables, por lo que permiten adaptarlos a las necesidades del momento y reconfigurarlos después en caso de ser necesario.
- Con la generación de alarmas y las herramientas de diagnóstico se puede localizar de forma rápida las fallas, esto permite reducir los paros productivos en las instalaciones y reduce de manera sustancial los costos de mantenimiento.
- Monitorización: representa los datos con severas restricciones de tiempo, de forma tal que permita a los operadores de la planta conocer el estado de los procesos en todo momento.
- Es posible operar remotamente con la mayoría de estos sistemas de control sobre los servicios o procesos productivos, la restauración tras una interrupción puede ser efectuada a distancia con mayor velocidad.
- El SCADA también ayuda a determinar la localidad de donde proviene la interrupción, la severidad de la misma y el número de personas afectadas, informaciones importantes para garantizar la asignación apropiada de los recursos necesarios para la recuperación. Sus reportes permiten analizar las causas de las diversas problemáticas.

Estas ventajas mencionadas hacen que la utilización de sistemas SCADA sea una forma más eficiente, rápida y segura de controlar la producción de una industria. La explotación de sus principales características, así como el uso homogéneo de los recursos humanos e informáticos que se manejan, lo hacen un potencial *software* para disímiles sistemas industriales o procesos específicos. El vertiginoso desarrollo de este tipo de producto en el mundo es notable, confirmando su eficacia. A continuación, se relacionan algunos de los más utilizados a nivel mundial.

1.1.1. Principales sistemas SCADA

❖ **PlantScape**

Es uno de los más utilizados actualmente a nivel mundial, el cuál fue desarrollado por Honeywell, este posee una herramienta de ingeniería del sistema, Control Builder, que incorpora una librería con más de 50 bloques de control analógico predefinidos. Estos bloques han sido diseñados a partir de la experiencia de Honeywell en sistemas DCS[3]. La documentación se realiza

[3] Sistema de Control Distribuido

utilizando el *Control Builder*, que la genera automáticamente. El lenguaje HTML [4] le permite obtener una ayuda particularizada al momento de la configuración. PlantScape está disponible con aplicaciones normalmente utilizadas, que incluyen control de calderas, análisis de disparo de planta, análisis predictivo y muchas más. Se puede añadir el RMPCT [5], para aplicaciones que requieran algoritmos de control más complejos (*[*5*]* y *[*7*])*.

Entre las características fundamentales que provee esta herramienta se destacan:

- Los cambios en la configuración pueden hacerse mientras el sistema está en línea.
- Posee una interfaz intuitiva.
- Selección de multiples objetos.
- Edición de multiples puntos.
- Informes normalizados.
- Facilidad en el manejo de alarmas.
- Fácil y rápida configuración de grupos y tendencias.
- Integra modelo común de información a la solución.

❖ **ARGOS-SCADA**

El proyecto Argos está siendo desarrollado para ser un SCADA con código abierto, así mismo, las herramientas que este proporciona actualmente sientan una base (con funcionalidades básicas) para implementar sistemas de supervisión en procesos automatizados.

ARGOS se ha diseñado con una arquitectura que permite adaptarse a los distintos esquemas de automatización moderna, en donde cada componente de *software* cuenta con estructuras de datos de alto rendimiento que operan de manera distribuida ya sea en una plataforma de red o en una misma computadora *[*5*]*.

❖ **InTouch**

[4] *Hypertext Mark up Language*
[5] *Robust Multivariable Predictive Control Technology* (Tecnología robusta y multivariable de control predictivo, en español)

Es un SCADA de la rama Wonderware [6] de la división de Administración de Operaciones de la compañía inglesa Invensys. Permite configurar alarmas y establecerles hasta 999 niveles de prioridad y hasta 8 niveles de jerarquía entre grupos de alarma con posibilidad de hasta 16 subgrupos para cada uno de ellos. Posibilita visualizar todas o un extracto de ellas de forma histórica y grabar en disco o imprimir en diferentes formatos personalizables. Las funciones de alarmas distribuidas incluyen reconocimiento global o selectivo, desplazamiento por la lista y visualización de alarmas procedentes de diferentes servidores en un único panel *[8]*.

Entre las características más significativas se destacan:

- Gráficos orientados a objetos.
- SuitLink / OPC: es un protocolo de comunicaciones elaborado por Wonderware de muy altas prestaciones para enlace de aplicaciones bajo TCP/IP o PROFIBUS, en el que se pueden configurar clientes OPC (OLE7 for Process Control).
- Gráficos de tendencia históricos y a tiempo reales.
- Lecturas y escrituras optimizadas.
- Posee un modelo de planta único que presenta una abstracción de su equipamiento físico y sistema.

❖ **EROS**

Desarrollado en 1993 e instalado por primera vez en la empresa niquelífera Ernesto Che Guevara, Moa. Es un sistema completamente cubano, diseñado y desarrollado en la División de Automatización, SERCONI *[51]*.

Entre sus principales características está que permite trabajar acoplado a diversos sistemas de colección de datos, como elemento único o formando parte de una red industrial. Tiene en cuenta todas las características de las variables medidas y realiza un potente tratamiento estadístico y determinístico de las mismas con solo configurarlo.

Su ambiente de trabajo es amistoso y la presentación agradable. Se configura de forma muy fácil (en línea) lo que permite que un usuario no especializado, con orientaciones sencillas, pueda configurarlo sin tener que depender de

[6] Es el líder mundial en la interfaz hombre-máquina, SCADA y en tiempo real de software de gestión de operaciones.
[7] *Object Linking and Embedding,* Vinculación e inclusión de objetos.

especialistas. Este sistema contiene la experiencia acumulada durante más de 18 años y ha sido validado en 67 plantas industriales.

Generalizando los principales sistemas SCADA se concluye que Intouch y PlantScape tienen como particularidad que son propietarios con un alto costo de adquisición, haciéndolos poco viables para nuestro país pues se hace imposible realizar análisis del código y/o cambiar características específicas para adaptarlos a nuestras condiciones, sin dejar de mencionar que generalmente estos vienen con *hardware* específicos para su funcionamiento que son difíciles de adquirir dado los asedios económicos que sufre nuestro país. Sin embargo, ambos presentan elementos del CIM que son aplicados en áreas específicas del sistema y en el caso de PlantScape es integrado como parte de la solución general para las soluciones de Honeywell. Por otra parte, ARGOS-SCADA y EROS son de código abierto, por lo que su despliegue resulta más eficaz para nuestras empresas, destacando que este último es de producción nacional altamente cualificado y con excelentes prestaciones.

1.2. Introducción al estándar CIM

El modelo CIM es un estándar descrito en UML [8] que organiza toda la información que puede ser necesaria en la gestión de los sistemas. Facilita la integración mediante un lenguaje común, basado en el estándar para habilitar estas aplicaciones o sistemas de acceso a datos públicos y el intercambio de información independientemente de cómo están estructurados internamente o de quién haya sido su fabricante. Lo que supone una reducción de costes y complejidad en los sistemas de gestión industrial. Esta norma debe ser entendida como una herramienta para permitir la integración en cualquier dominio donde se necesita un modelo de sistema de alimentación común para facilitar la interoperabilidad y compatibilidad del enlace entre las aplicaciones y los sistemas independientes de cualquier aplicación particular.

El CIM viene descrito en las normas IEC[9] mediante diagramas de clases y diagramas de paquetes UML. Por este motivo en el siguiente acápite se proporcionarán unos conceptos y características básicos para facilitar la interpretación de estos diagramas.

1.2.1. Generalidades y uso del estándar CIM

[8] *Unified Modeling Language*, por sus siglas en inglés.
[9] *Electric Power Research Institute, por sus siglas en inglés*

El UML [10] es un lenguaje muy extendido en el mundo informático de la actualidad, que permite representar todo tipo de sistemas mediante el empleo de notación gráfica. En el grupo de diagramas que lo conforman se encuentran los diagramas de clases y de paquetes, que son los dos tipos de diagramas que se emplean en la norma IEC para describir el modelo CIM.

El modelo CIM se define dentro de una serie de normas de la IEC, entre las que se encuentran:

- IEC 61850 *Communication Networks and Systems in Substations*, especializado en la comunicación del *hardware* y el *software* en las subestaciones.
- IEC 61970 centrado en los sistemas de gestión de redes de transportes, también conocidos como EMS[11].
- IEC 61968 centrado en los sistemas de gestión de las redes de distribución, conocidos como DMS[12].

Estas normas no solo definen el modelo CIM, sino que además describen una arquitectura de referencia para la integración entre las aplicaciones que forman parte de un sistema de gestión y los requisitos generales de las mismas, especifican los mecanismos (servicios) de comunicación por los cuáles se puede acceder a la información de una aplicación y finalmente especifican formatos (como el CIM/XML) empleados para el intercambio de información.

Estructura del estándar IEC 61850 por capas *[13]*.

[10] *Unified Modeling Language.* Disponible en: http://www.uml.org/.
[11] *Energy Management System, por sus siglas en inglés*
[12] *Distribution Management System, por sus siglas en inglés*

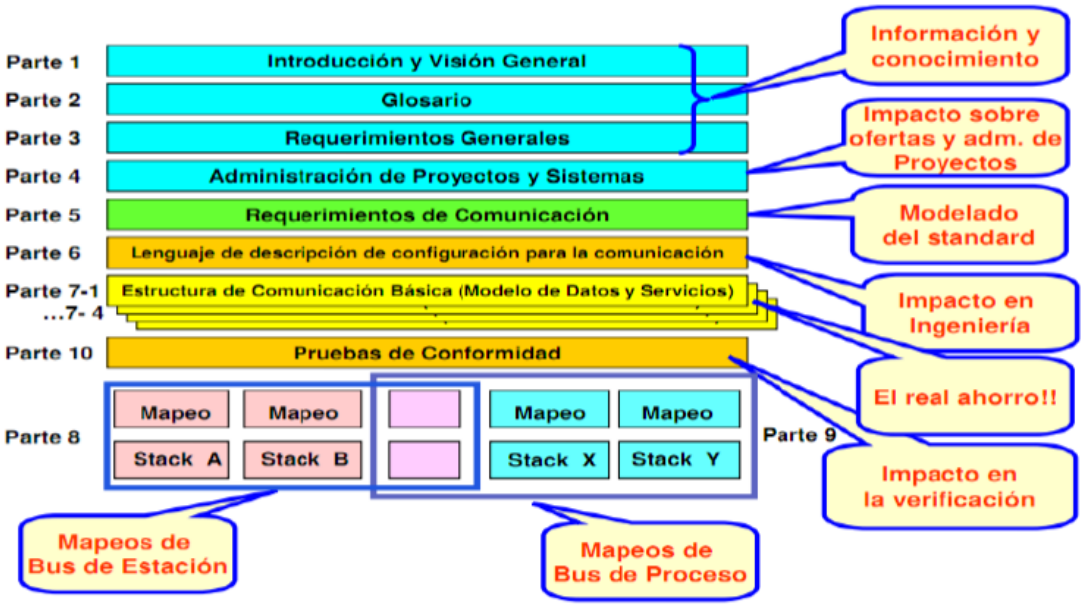


Figura 2: Partes del estándar IEC 61850.

Parte 1: define las formas de comunicación entre IEDs [13] en la subestación eléctrica y los requisitos relacionados al sistema.

Parte 2: contiene el glosario de terminología específica y de definiciones que se utilizan en el contexto de los sistemas automáticos en subestaciones eléctricas, así como en las normas complementarias a esta.

Parte 3: especifica los requisitos generales de las redes de comunicación, con especial énfasis en los requisitos de calidad. Igualmente trata los requisitos relativos a las condiciones ambientales y de servicios auxiliares, así como las recomendaciones correspondientes a requisitos específicos provenientes de otras normas y especificaciones.

Parte 4: describe los requisitos del proceso de gestión de sistema y proyecto y de las herramientas de soporte especial para ingeniería y pruebas.

Parte 5: describe los requisitos para la comunicación de funciones y modelos de los equipos eléctricos en las subestaciones.

Parte 6: especifica una descripción del lenguaje para la configuración de los IEDs en las subestaciones eléctricas.

[13] *Intelligent Electronic Device, por sus siglas en inglés*

Parte 7: se definen las estructuras básicas de comunicación para los equipos eléctricos en subestaciones y alimentadores de distribución.

Parte 8: define el mapeo específico de servicio de comunicación de los objetos y servicios de la Interfaz de Servicios de Comunicación Abstracta a especificación de mensajes de fabricación.

Parte 9: define el mapeo específico de servicio de comunicación para la comunicación entre el nivel campo (bahía) y el nivel de proceso en enlaces punto a punto.

Parte 10: define los métodos de pruebas de conformidad del equipo usado en las subestaciones eléctricas.

Como todo modelo orientado a objetos de gran tamaño, el CIM agrupa a todas sus clases en distintos paquetes. La figura 3 muestra el diagrama UML en el que se representan los paquetes que se incluían en las primeras versiones de la IEC 61970-301. Las flechas que unen a dos paquetes indican que existe dependencia entre ellos, es decir, una modificación en uno de los paquetes puede propiciar cambios en el otro.

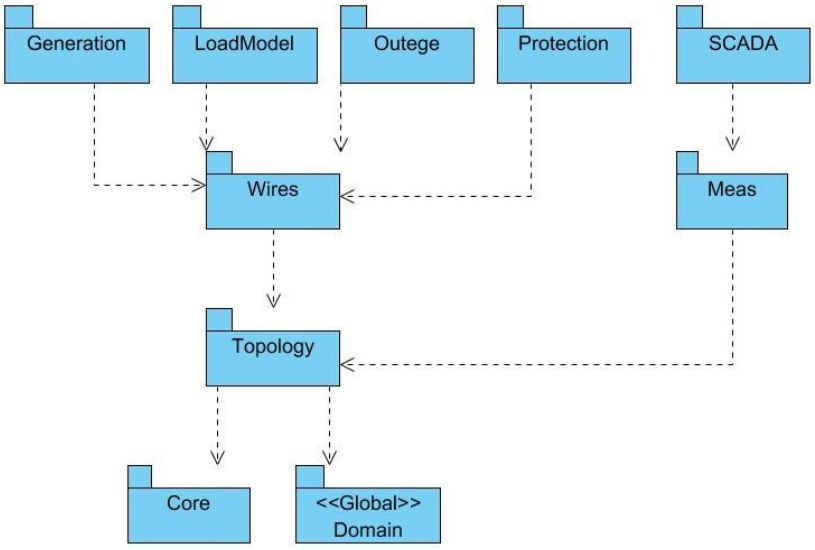


Figura 3: Diagrama de Paquetes UML de la IEC 61970-301.

Para lograr un mejor entendimiento del diagrama, a continuación, se describen brevemente los paquetes que lo conforman:

- En el paquete ***Wires*** se encuentran las clases que representan los equipos físicos que pueden formar parte de un sistema de potencia. Un ejemplo de clase incluida en este paquete es *Breaker*, que representa los interruptores.

- El paquete ***Topology*** proporciona las clases necesarias para poder describir las conexiones entre los equipos. Así define, entre otras, la clase *ConnectivityNode*.
- El paquete central ***Core*** incluye una serie de clases, como *IdentifiedObject*, que serán padres (superclases) de la mayoría del resto de las clases del modelo. Contiene el núcleo y entidades, compartida por las aplicaciones más comunes de las colecciones de estas entidades.
- ***Generation*** incluye las clases necesarias para representar la información referente a la generación de la energía eléctrica, como, por ejemplo: *GenerationUnit* o *HydroTurbine*.
- ***Meas*** comprende casi todas las clases relacionadas con la representación de las medidas en los sistemas: *Measurement, MeasurementType,* etc.
- ***LoadModel*** incluye las clases necesarias para describir todo lo referente al consumo de energía eléctrica. Contiene, entre otras cosas, a la clase *EnergyConsumer*, que permite representar todo tipo de consumidores de energía eléctrica.
- ***Outage*** permite describir programas de planificación para la operación y mantenimiento de los sistemas de potencia para la prevención y actuación ente apagones. Incluye, entre otras, la clase *OutageSchedule*, que permite describir la información acerca de los períodos programados en los que un equipo determinado está fuera de servicio, por mantenimiento o realización de algún test.
- ***Protection*** proporciona las clases necesarias para representar los equipos de protección (clase *ProtectionEquipment*) y las secuencias de cierre automático (clase *RecloseSequence*).
- ***SCADA*** incluye, entre otras, la clase *RemoleUnit*, que representa los dispositivos electrónicos situados en la instalación y que intercambian información con la aplicación SCADA localizada en el sistema de gestión. También permite describir los enlaces de comunicación con estas unidades remotas mediante la clase *CommunicationLink*.
- ***Domain*** es el paquete que incluye las definiciones de todos los tipos de datos que se asignan a los valores de los atributos descritos en el resto del modelo: desde tipos primitivos como *float*, hasta tipos como *Temperature*, por ejemplo.

En la actualidad, el modelo CIM es más amplio. Así, en la norma IEC 61968-11 (para sistemas DMS) se añaden nuevos paquetes al modelo original mediante un proceso de extensión basado en la necesidad de agregar elementos de control que satisfagan todas las necesidades para una idónea gestión industrial. Dicha extensión se logra mediante la creación de nuevas clases derivadas de otras ya existentes, acción que permite aplicar el CIM en instalaciones con ciertas particularidades.

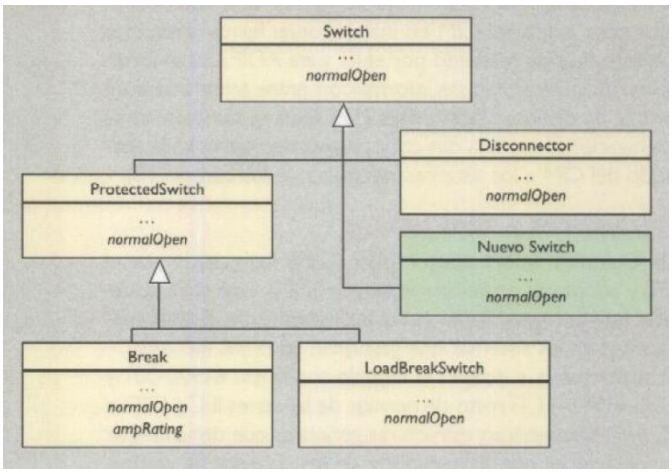


Figura 4: Ejemplo de extensión del modelo CIM.

La creciente solicitud de tecnología ha conllevado a una gran demanda de manejo de información, así como comunicación y almacenamiento de datos por lo que la competencia de industrias desarrolladoras de *software* y *hardware* para estaciones eléctricas crece cada vez más y se ve la necesidad de crear un modelo común para todas estas tecnologías en busca de integración, flexibilidad y apertura hacia el futuro, logrado por el estándar IEC61970. Orientado a lograr homogeneidad y mejora en el rendimiento del SCADA SAINUX como solución informática, se propone la incorporación de este modelo al desarrollo del mismo con el fin de elevar los resultados y la calidad del producto.

1.3. Módulo HMI de un SCADA

En un proceso industrial, cualquier proceso que se desee controlar y automatizar tiene que ser monitorizado. Existen ocasiones en las que se debe censar temperatura, presión, posición, flujo, velocidad, proximidad, etc., estas actividades deben ser realizadas con elementos de medición los cuáles se les denomina sensores. El estado de los sensores constituye las entradas para el elemento controlado, como puede ser un PLC, PAC, DAQ, PC, entre otros.

Estos controladores, según la entrada del sensor que reciban, van a tomar alguna acción que tengan previamente programada, mediante los actuadores. Los actuadores son dispositivos de control que básicamente interactúan con el proceso a controlar, como lo son los relevadores, arrancadores, electroválvulas, etc. *[21]*.

Hasta este punto se tiene un sistema automatizado basado en controladores, actuadores y sensores. Pero dicho sistema está aislado del elemento humano; claro que en la mayoría de los casos esto es lo que se necesita, pues el controlador es un dispositivo con capacidades superiores a las de un ser humano a la hora de realizar tareas de mucha precisión, repetitivas y de manera ininterrumpida, pero es el hombre quien toma decisiones en situaciones que implican mayor flexibilidad, sentido común o simplemente el proceso es tal que antes de iniciarlo los operadores deben introducir informaciones al sistema, como puede ser el material o sustancias que se van a usar en una etapa del proceso, la cantidad de producto, la disponibilidad de los contenedores o bien detener el proceso por alguna razón. Además, los operadores del proceso necesitan interactuar con el sistema de control de manera intuitiva, fácil de conocer y que les proporcione la suficiente información del proceso en el momento que se necesite. Por esta razón se crean las llamadas Interfaz Hombre-Máquina (HMI).

Un HMI es una interfaz gráfica, muy simple, que muestra información del proceso en tiempo real, utilizando diagramas, esquemas, algunos contornos y hasta animaciones en pantallas, entonces se puede decir que un HMI es la interfaz visual que muestra el estado del sistema y los componentes que lo integran y con el que interactúa el operador para monitorear y controlar los recursos.

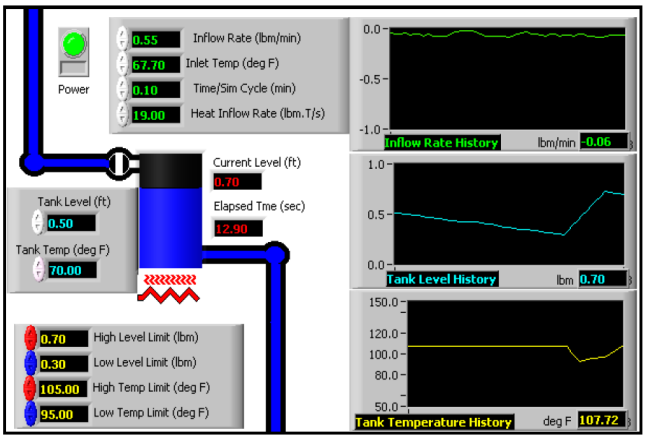


Figura 5: Interfaz HMI en LabVIEW (tomado de los ejemplos de labVIEW).

Las principales funciones de una interfaz de usuario son:

- Puesta en marcha y apagado.
- Control de las funciones manipuladoras del equipo.
- Manipulación de archivos y directorios.
- Herramientas de desarrollo de aplicaciones.
- Comunicación con otros sistemas.
- Información de estado.
- Configuración de la propia interfaz y entorno.
- Intercambio de datos entre aplicaciones.
- Control de acceso.
- Sistema de ayuda interactivo.

1.4. Herramientas y tecnologías

A continuación, se argumentará sobre las tecnologías y herramientas utilizadas en el desarrollo del *software*, que en este caso no son objeto de selección, pues las mismas fueron analizadas y evaluadas desde los inicios del proyecto SCADA SAINUX.

1.4.1. Lenguaje y herramienta de modelado

UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de *software*. Prescribe un conjunto de notaciones y diagramas estándares para modelar sistemas orientados a objetos, y describir la semántica esencial de lo que estos diagramas y símbolos significan. UML se puede usar para modelar distintos

tipos de sistemas: sistemas de *software*, sistemas de *hardware* y organizaciones del mundo real *[43]*.

Visual Paradigm es una herramienta de Ingeniería de Software Asistida por Ordenador (CASE), desarrollada por la compañía Visual Paradigm International, que utiliza UML como lenguaje de modelado. Permite soportar el ciclo de vida completo del *software*: análisis, diseño, implementación y despliegue, la captura de requisitos, el dibujo de diagramas UML, la realización de ingeniería inversa y generación de código C++. Puede ser integrado con varias herramientas como son: Eclipse/IBM, Builder, Net Beans IDE, Oracle JDeveloper, BEA Weblogic. Está disponible en varias ediciones, cada una destinada a necesidades específicas: Enterprise, Professional, Community, Standard, Modeler y Personal *[43]*.

La versión empleada en la solución es el Visual Paradigm 5.0, lanzada el 16 de agosto del 2010.

1.4.2. Entorno y marco de trabajo

QtCreator ofrece un ambiente de desarrollo completo para la creación de aplicaciones Qt. Es una herramienta ligera y multiplataforma, con un enfoque estricto hacia las necesidades de los desarrolladores de aplicaciones Qt.

Las principales características son *[9]:*

- El editor avanzado de C++, para escribir, editar y navegar por el código fuente sin utilizar el ratón (*mouse*).
- Interfaz gráfica para la depuración, que incrementa la percepción de la estructura de clases de Qt.
- Integración con QtDesigner para editar los archivos de interfaz gráfica de usuario.
- La herramienta Localizador (*Locator*) para la navegación rápida por archivos de proyecto, funciones, clases e informaciones de ayuda.
- Integración con QtAssistant, facilitando el acceso a la ayuda de la API Qt (tipos de datos, funciones) de Qt a través de una ayuda sensible al contexto.
- Construir y ejecutar proyectos Qt con la herramienta multiplataforma qmake.

Qt es un marco de trabajo (*framework*) escrito en C++ para el desarrollo de aplicaciones de Interfaz Gráfica de Usuario, el cual sigue la filosofía de *software* "escriba una vez, compile donde quiera" (*write once, compile anywhere*). Además, amplía las posibilidades del lenguaje C++ con una extensa biblioteca de clases, de uso intuitivo y dividido en módulos, en los cuales se puede encontrar desde programación de interfaces gráficas de usuarios hasta programación de redes, procesamiento de textos enriquecidos, acceso a datos almacenados en varios de los gestores de bases de datos más populares. Incorpora herramientas para escribir aplicaciones robustas y en el menor tiempo posible como el Diseñador Qt (*QtDesigner*). A esto se le suma su extensa base de datos de ejemplos listos para usar.

Qt está disponible bajo licencia LGPL[14], permitiendo el desarrollo de Software Libre, y si se desea, *software* comercial no libre, en ambos casos sin vernos obligados al pago de licencias o derechos *[9]*.

1.5. Metodología de desarrollo de software

El objetivo principal que busca la ingeniería de *software* es convertir el desarrollo de *software* en un proceso formal, con resultados predecibles, que permitan obtener un producto final de alta calidad y satisfaga las necesidades y expectativas del cliente. La dificultad propia de los nuevos sistemas, y su impacto en las organizaciones, ponen de manifiesto las ventajas, y en muchos casos la necesidad, de aplicar una metodología formal para llevar a cabo los proyectos de este tipo.

Una metodología impone un proceso de forma disciplinada sobre el desarrollo de *software* con el objetivo de hacerlo más predecible y eficiente. Una metodología define una representación que permite facilitar la manipulación de modelos, y la comunicación e intercambio de información entre todas las partes involucradas en la construcción de un sistema *[33]*.

Existen diferentes modelos y metodologías que han sido, en los últimos años, herramientas de apoyo para el desarrollo del *software.* Pero, ¿qué es una metodología de desarrollo de *software*?, es un enfoque estructurado y metódico para el desarrollo de *software* que incluye modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos.

[14] Licencia Pública General de GNU (*GNU General Public License*, por sus siglas en inglés) es la licencia más ampliamente usada en el mundo del software y garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el *software.*

Mundialmente se clasifican en tres grandes grupos: ágiles, robustas e híbridas. Dentro de esta última se incluye la metodología AUP-UCI[15].

En la UCI para garantizar un mejor acoplamiento en la producción se les realizaron cambios a las fases descritas en la metodología AUP original quedando de la siguiente forma.

Descripción de las fases del ciclo de desarrollo con la variación UCI *[9]:*

- **Inicio:** durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
- **Ejecución:** en esta fase se ejecutan las actividades requeridas para desarrollar el *software,* incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elabora la arquitectura, el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del *software*.
- **Cierre:** en esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

1.5.1. Conclusiones parciales de metodologías

Siguiendo los métodos de investigación especificados en la introducción del presente trabajo se puede sintetizar que el modelo del Proceso de Desarrollo de Software (PDS) con el que se está trabajando en el SCADA SAINUX es prescriptivo pues se tienen un conjunto de elementos que describen los flujos de trabajo, las actividades y la interrelación entre los componentes. Entre los distintos tipos de modelos prescriptivos del PDS se ajusta más al especializado, específicamente a la Programación Orientada a Aspectos que provee un proceso para definir, especificar, diseñar y construir aspectos de

[15]*Agile Unified Process*, Proceso Unificado Ágil con variante UCI.

software como interfaces, seguridad y gestión de memoria que impactan varias partes del sistema en desarrollo.

La metodología del PDS a seguir es AUP, una metodología híbrida que recoge características esenciales de RUP como el modelado en UML de las clases y los flujos de información, aspecto fundamental para este trabajo, pero además se necesita que se valore el tamaño del equipo de trabajo, la criticidad del *software*, la cultura y el dinamismo.

AUP-UCI logra recoger las características esenciales de la solución y dentro de su marco de trabajo garantiza que se pueda ajustar a las especificaciones del proyecto. Además, su selección se basa en la política de calidad y desarrollo de *software* de la universidad y el requisito descrito por el cliente.

Conclusiones Parciales

Luego de realizar el análisis del estado del arte y confeccionar el marco teórico de la investigación referente a los sistemas SCADA y en estrecha relación con componentes y conceptos básicos imprescindible para el análisis y comprensión de la propuesta de solución se puede concluir que:

- Los sistemas SCADA son software que manejan un alto flujo de información referente al estado y funcionamiento industrial, los cuáles vinculas desde componentes de *hardware* (sensores, estaciones de trabajo, válvulas, etc.), *software* (módulos, datos, drivers, aplicaciones, etc.) hasta mecanismos de procesos y gestión del negocio.
- La creciente demanda de estos sistemas, para garantizar un óptimo funcionamiento de las industrias, así como para obtener el máximo rendimiento de cualquier negocio industrial, hacen de los SCADA una solución viable en el mercado nacional e internacional pues resuelven problemas generados por la intervención humana, así como condiciones del entorno que lo imposibilitan.
- Los SCADA son sistemas que se conforman por diferentes módulos, los cuales necesitan de una estandarización para su comunicación pues sino se generan deficiencias como las planteadas en la situación problémica. Por ellos surgió el estándar CIM como modelo de comunicación aprobado por entidades patentadas que se aplicarán como propuesta de solución al SCADA SAINUX.

- Para dar cumplimiento al diseño e implementación de dicho estándar, propuesto a desarrollar en las tareas y en función del objetivo planteado, se definió el uso de un conjunto de herramientas, tecnologías y metodologías:
  - ✓ Framework: Qt
  - ✓ IDE: QtCreator
  - ✓ Leguaje de modelado: UML
  - ✓ Herramienta de modelado: Visual Paradimg
  - ✓ Lenguaje de programación: C++
  - ✓ Sistema Operativo: GNU/Linux
  - ✓ Metodología: AUP

Los siguientes capítulos estarán orientados a dar solución al problema de investigación en pos de alcanzar el objetivo propuesto para tributar a los posibles resultados.

CAPÍTULO 2: "ANÁLISIS DEL Y DISEÑO DE LA SOLUCIÓN"

2. Introducción

En el presente capítulo se exponen los requisitos funcionales y no funcionales, así como las características actuales del sistema, los principales elementos de la arquitectura de datos del modelo CIM y del *software* para realizar una propuesta de solución a los problemas detectados basados en los diagramas y patrones documentados.

2.1. Especificaciones del estándar CIM del SCADA SAINUX

Para una mejor gestión de la información y los recursos de una industria, fue creado el Modelo Común de Información. Este estándar define y especifica cómo se relaciona cada componente (clase) y el flujo de información necesaria para cumplir con los más exigentes requisitos de un sistema. Actualmente en el SCADA SAINUX existe este modelo, pero presenta algunos inconvenientes que imposibilita obtener un óptimo rendimiento.

Se han analizado cuatro paquetes (*Security, Acquisition, Historical, HMI*), cada uno contiene una serie de clases concretas, que definen el funcionamiento integral del sistema. Durante la revisión se detectaron irregularidades como la existencia de información no relacionada al modelo de datos, alta concurrencia de clases, relaciones ineficientes, baja escalabilidad y débil seguridad en el acceso a los recursos potenciales. Como resultado de los problemas descritos el *software* presenta poca mantenibilidad por lo que se concluye que se debería generar una nueva propuesta del modelo para reducir el costo y tiempo de trabajo.

2.2. Introducción a la Ingeniería de Requisitos

La Ingeniería de Requisitos cumple un papel primordial en el proceso de producción de *software*, ya que enfoca un área fundamental: la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema; de esta manera, se pretende minimizar los problemas relacionados al desarrollo de sistemas.

Los principales beneficios que se obtienen de la Ingeniería de Requerimientos son:

- Permite gestionar las necesidades del proyecto en forma estructurada.
- Mejora la capacidad de predecir cronogramas de proyectos, así como sus resultados.
- Disminuye los costos y retrasos del proyecto.
- Mejora la calidad del software.
- Mejora la comunicación entre equipos, especialmente entre clientes y desarrolladores.
- Evita rechazos de usuarios finales.

Las estrategias recomendadas para la especificación de los requisitos del *software* están descritas por IEEE 830-1998[16]. Este estándar describe las posibles estructuras, los contenidos deseables, y calidades de una especificación de requisitos del *software*.

Los requisitos se dividen en dos grupos:

- **Funcionales:** son los que el usuario necesita que efectúe el *software*. Normalmente se identifican como los requisitos que responden a la pregunta ¿qué hace? Ej: El sistema debe emitir una alerta al crear un nuevo proyecto.
- **No funcionales:** son los "recursos" para que trabaje el sistema de información (redes, tecnología). Ej: el soporte de almacenamiento a usar debe ser MySQL. Normalmente se identifican como los requisitos que responden a la pregunta ¿cómo lo hace? Ej: rápido, fácil etc.

2.2.1. Requisitos funcionales del SCADA SAINUX

Los requisitos funcionales detectados con el análisis del sistema y los propuestos por el cliente que se necesita que efectúe el *software* son:

RF 1: Gestionar los datos del modelo para el paquete *Security.*

RF 2: Gestionar los datos del modelo para el paquete *Acquisition.*

RF 3: Gestionar los datos del modelo para el paquete *Principal.*

[16] IEEE 830-1998. Disponible en: http://www.ctr.unican.esasignaturasi/s1/IEEE830_esp.pdf

2.2.2. Requisitos no funcionales del SCADA SAINUX

Los requisitos no funcionales del SCADA SAINUX responden a lo que se necesita y cómo se hace para que trabaje el sistema de información. A continuación se relacionan subdivididos por tipo:

- **Rendimiento**

  RNF 1: Se deben minimizar los tiempos de respuesta de solicitudes al modelo, tanto de servicios como de información, por parte de otros recursos (clases o usuarios).

- **Estabilidad**

  RNF 2: Se espera la capacidad del modelo para minimizar los efectos inesperados de las modificaciones del software.

- **Mantenibilidad**

  RNF 3: Debe presentar la capacidad de ser modificado. Las modificaciones pueden incluir las correcciones, mejoras o adaptación del *software* a los cambios en el ambiente, y en los requisitos y las especificaciones funcionales.

2.3. Modelo de dominio

Un modelo del dominio se utiliza con frecuencia como fuente de inspiración para el diseño de los objetos *software*. El modelo del dominio muestra (a los modeladores) clases conceptuales significativas en un dominio de problema; es el artefacto más importante que se crea durante el análisis orientado a objetos *[45]*.

Haciendo uso del lenguaje UML se puede modelar un problema donde se necesite conceptualizar gráficamente elementos relacionales entre entidades como son:

- Objetos del dominio o clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

Para lograr una mejor comprensión de los diagramas del modelo de dominio, por su gran amplitud y complejidad, se dividieron en paquetes. Los cinco

paquetes presentados agrupan un conjunto de clases que resumen el funcionamiento de un módulo en específico de acuerdo a su concepción lógica.

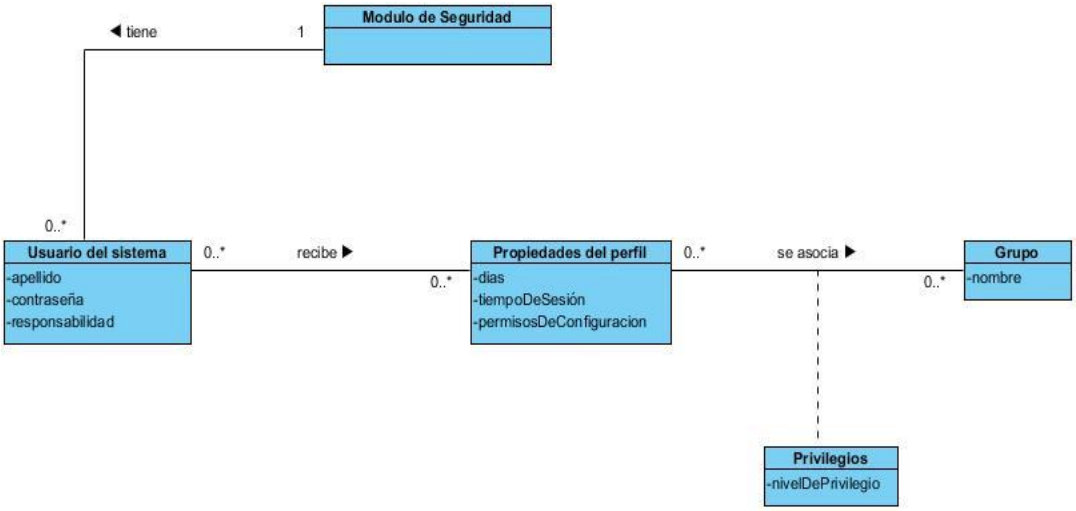


Figura 6: Modelo de dominio del paquete Security

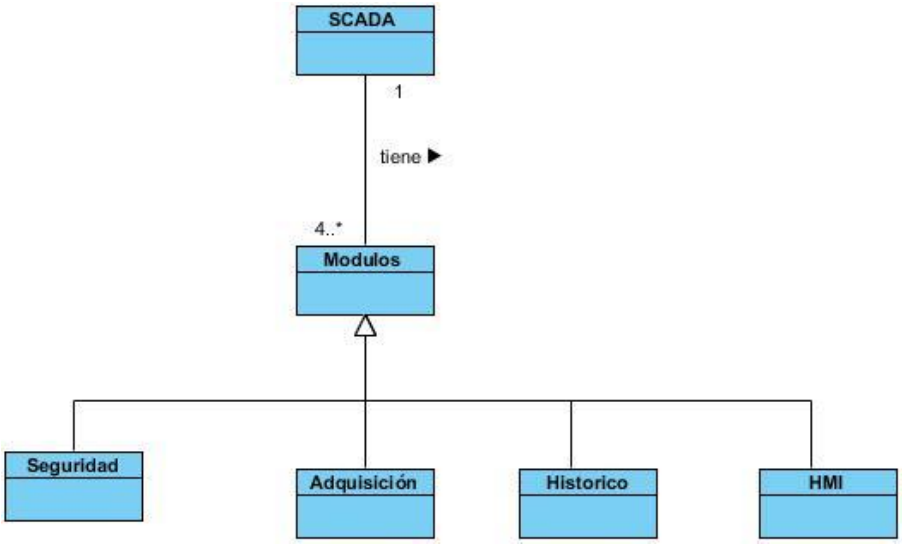


Figura 7: Modelo de dominio del paquete CIM.

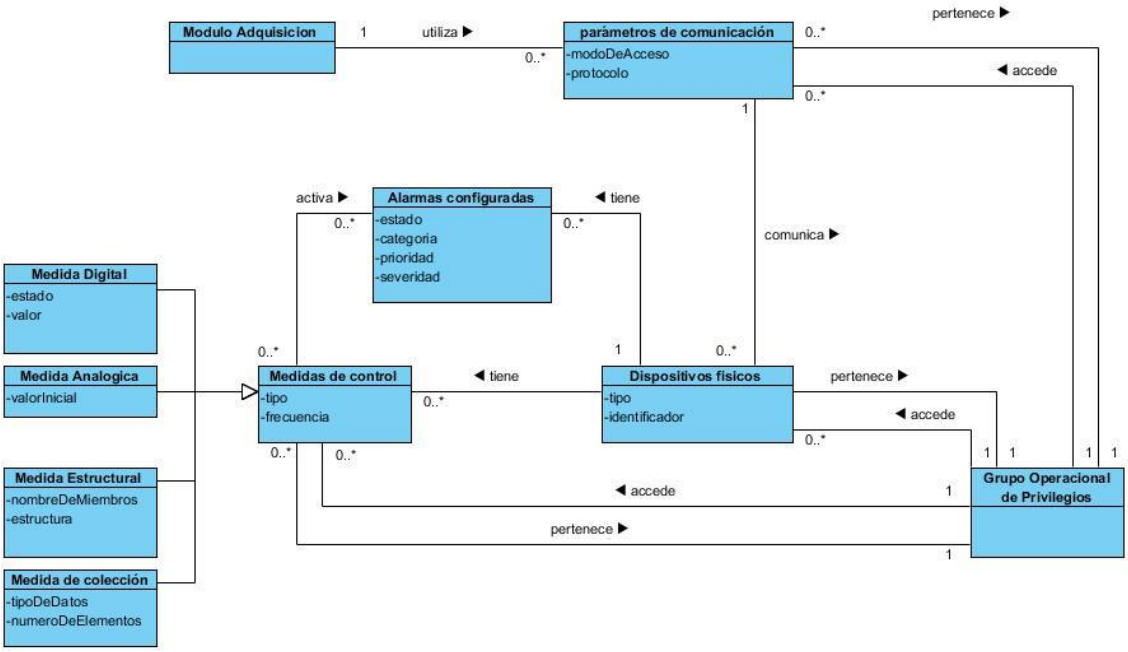


Figura 8: Modelo de dominio del paquete Acquisition.

2.4. Arquitectura de la propuesta de solución

La arquitectura de un *software* es el diseño de más alto nivel de la estructura de un sistema. Consiste en definir y aplicar un conjunto de patrones e ideas abstractas que proporcionan un marco de trabajo o de desarrollo con el objetivo de dar cumplimiento a las restricciones del ambiente de ejecución del *software* o las redactadas por el cliente. Además, se definen los componentes que llevan a cabo alguna tarea de cómputo, sus interfaces y la comunicación entre ellos. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar el *hardware* que dará soporte al sistema, por lo que se puede concluir que es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad.

Específicamente, un estilo arquitectónico determina el vocabulario de componentes y conectores que puede ser usado, así como un conjunto de restricciones de cómo pueden ser combinados. Particularmente el **estilo arquitectónico orientado a objetos** es donde los componentes se basan en los principios orientados a objetos (OO). En una forma simple, un diseño orientado a objetos permite diseñar sistemas para encapsular los datos y los objetos proveen explícitamente interfaces a otros objetos; las interfaces están separadas de las implementaciones. Pero la principal característica de este

estilo es que se puede modificar la implementación de un objeto sin afectar la interfaz. Hay muchas variantes en este estilo ya que algunos sistemas permiten que los objetos sean tareas concurrentes y otros admiten que los objetos contengan diferentes interfaces.

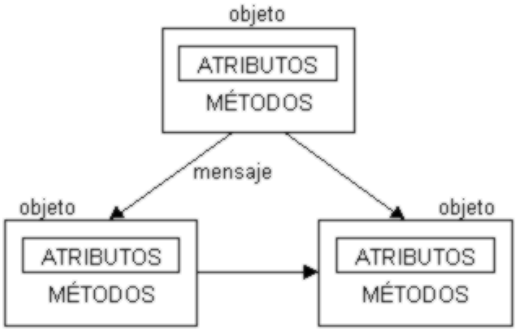


Figura 9: Ejemplo de modelo con arquitectura orientada a objetos

Entre las principales características de esta arquitectura está que los componentes de este estilo son los objetos o más bien instancias de los tipos de datos abstractos, donde la comunicación y coordinación entre ellos se realiza a través del paso de mensajes garantizando que se pueda modificar la implementación de un objeto sin afectar a sus clientes pues se maneja la filosofía de que un objeto es, ante todo, "*una entidad reutilizable en el entorno de desarrollo".* Sin embarco cabe destacar que presenta como desventaja que, para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.

El modelo propuesto presenta una estructura jerárquica donde los objetos (clases) representan una abstracción del modelo de dominio. Además, hace uso de patrones de diseños comunes en este tipo de arquitectura como es *Builder, Composite*y *Observer*. A continuación, se muestra el modelo con las características antes descritas que dan confirmación de la existencia del estilo arquitectónico propuesto:

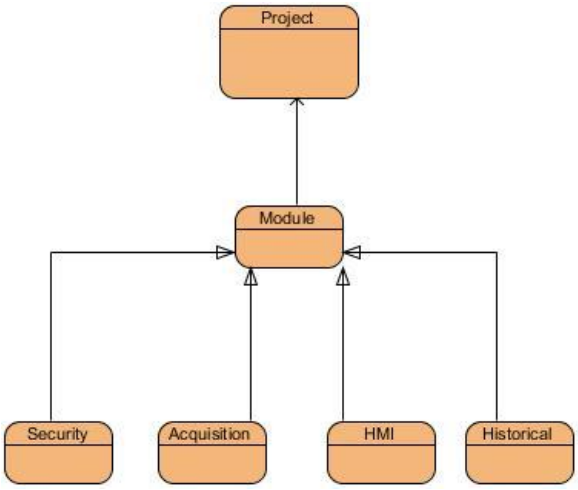


Figura 10: Modelo de datos CIM de la solución con arquitectura orientada a objetos.

2.5. Patrones de diseño

Un patrón de diseño es una buena práctica documentada de una solución que ha sido aplicada exitosamente en múltiples ambientes para resolver problemas que ocurren en un conjunto de situaciones. Puede ser visto como una encapsulación de solución reusable que ha sido aplicada para resolver un problema común en el diseño y también puede ser visto como una expresión documentada de resolver un problema que es observado o descubierto durante el estudio o construcción de un *software [24]*.

En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de *software* que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

Estos patrones contribuyen a reutilizar diseño gráfico, identificando aspectos claves de la estructura de un diseño que puede ser aplicado en una gran cantidad de situaciones. La importancia de la reutilización del diseño no es despreciable, ya que ésta nos provee de numerosas ventajas: reduce los esfuerzos de desarrollo y mantenimiento, mejora la seguridad informática, eficiencia y consistencia de nuestros diseños, y nos proporciona un considerable ahorro en la inversión. Además, mejoran la flexibilidad, modularidad y extensibilidad, factores internos e íntimamente relacionados con la calidad percibida por el usuario.

Por la gran variedad que existe y sus múltiples finalidades, estos patrones se han dividido en dos grandes familias conocidas como GRASP[17] y GoF[18].

2.5.1. Patrones de diseño GRASP

Una de las cosas más complicadas en Orientación a Objeto consiste en elegir las clases adecuadas y decidir cómo estas clases deben interactuar. Incluso cuando utilizamos metodologías rápidas como programación extrema (*extreme programming*) y centramos el proceso en el desarrollo continuo, es inevitable elegir cuidadosamente las responsabilidades de cada clase en la primera codificación y, fundamentalmente, en la refactorización de nuestro programa.

Por lo tanto, podemos decir que los patrones GRASP son una ayuda de aprendizaje que permiten al desarrollador a entender lo esencial del diseño de objetos y a aplicar el razonamiento del mismo de una forma metódica, racional y explicable. Este enfoque se entiende en el uso de los principios del diseño basado en patrones para la asignación de responsabilidades a las clases y objetos de una aplicación *[26]*.

Los principales patrones que describe Cuevas *[60]* en su literatura y son reconocidos por la comunidad informática que además son empleados en la solución son:

- **Creador:** el patrón creador ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización, aportando bajo acoplamiento, mayor claridad, encapsulación y reutilización del diseño realizado. Está presente en clases como *Security* que cuenta con los métodos ***addAssociationUserProfile()*** y ***addAssociationGroupProfile()***, funciones que instancian objetos del tipo de la clase asociada a ella.

[17] Acrónimo de *General Responsibility Assignment Software Patterns* y son patrones generales de *software* para asignación de responsabilidades.
[18] Abreviación del grupo *Gang of Four*, este grupo se encarga de describir patrones de diseño en el libro "*Desing Patterns*".

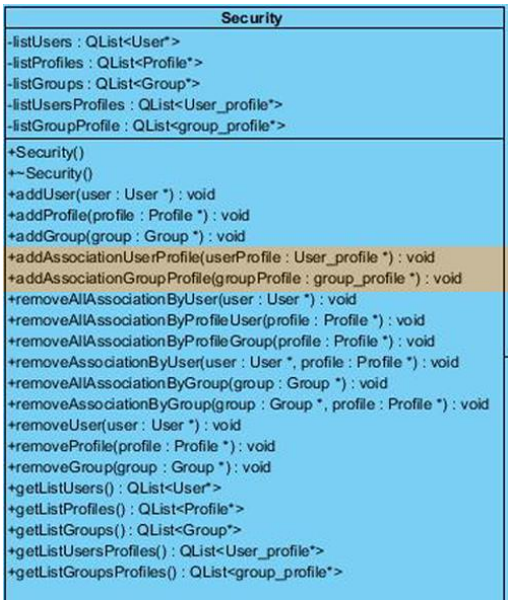


Figura 11: Ejemplo de patrón creador en el diagrama del paquete Security

- **Experto:** asigna la responsabilidad al experto en la información. Indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtiene un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento). Durante todo el diagrama se manejó la información por cada entidad que le corresponde y se le asignó la responsabilidad de acuerdo a esa información por lo que se puede afirmar que el patrón está presente en todas las clases.

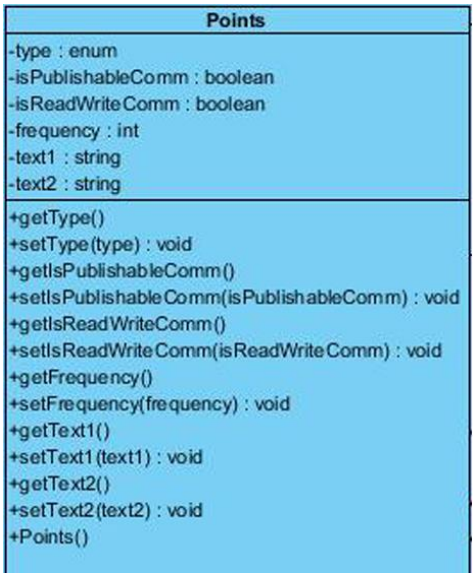


Figura 12: Ejemplo de patrón experto en el diagrama del paquete Acquisition

- **Alta cohesión:** indica que la información que almacena una clase debe de ser coherente y debe estar relacionada con la clase facilitando la solución al problema ¿Cómo mantener manejable la complejidad? El patrón está presente en todo el diagrama debido a que cada clase solo almacena la información correspondiente a ella misma.
- **Bajo acoplamiento**: es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. Facilita la solución al problema ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización? Reflejada en todo el diagrama.

2.5.2. Patrones de diseño GoF

En la bibliografía se propone un total de 23 patrones que pueden ser utilizados para el correcto diseño de un *software*, estos patrones son reconocidos como los patrones GoF.

El uso de patrones en el diseño de *software* o estándares proporcionan una estructura conocida por todos los programadores, de manera que la forma de trabajar no resulte distinta para cada uno de ellos. Así la incorporación de un nuevo programador, no requerirá conocimiento de lo realizado anteriormente por otro. Permiten tener una estructura de código común a todos los proyectos que implemente una funcionalidad genérica. Su utilización, permite ahorrar grandes cantidades de tiempo en la construcción de *software*.

Para mejorar la comprensión de cada uno de ellos a continuación se describe la finalidad particular de estos patrones:

- ***Builder:*** separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones. Este patrón está presente en el paquete Acquisition con la construcción de puntos (*Point*) y sus diferentes tipos (*array, structure, analog, digital*) separado para especificar sus características, disminuir la complejidad y facilitar la creación de varias instancias de puntos.
- ***Composite:*** combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera

uniforme a los objetos individuales y a los compuestos. Está presente en la estructura de árbol del proyecto, ejemplo: ***Project -> Module -> Security, HMI, Historical, Acquisition.***

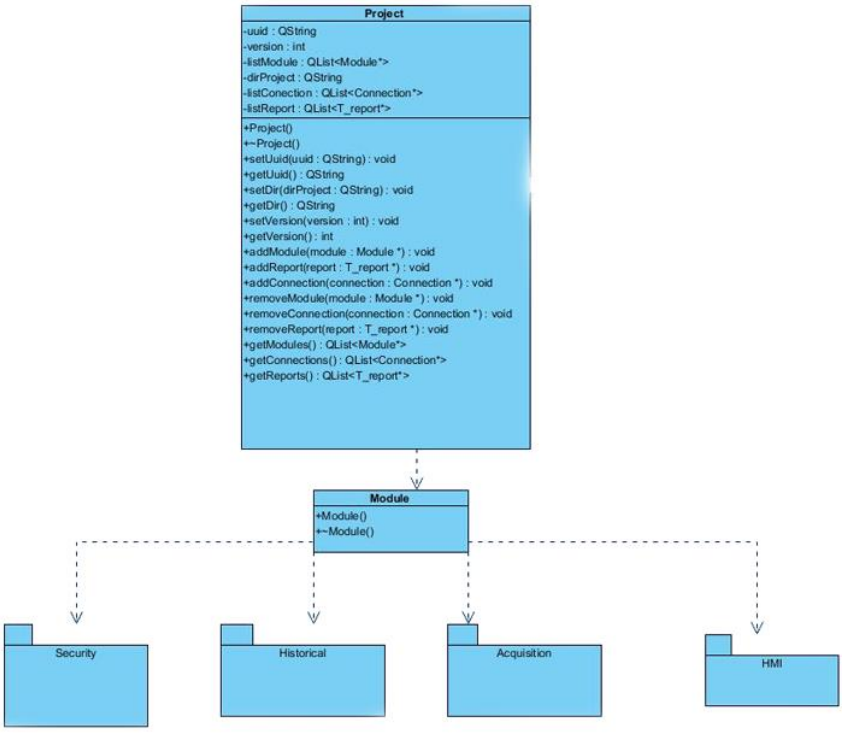


Figura 13: Ejemplo de patrón compositer en el diagrama del paquete Principal

- ***Iterator*:** proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna. Presente en el diagrama con la clase ***Module*** que representa una interfaz para los módulos específicos *Security*, *Historical, Acquisition* y *HMI.* Los métodos que contenga representará las funciones abstractas que manejarán cada uno de ellos facilitando el trabajo con los iteradores presentes en todo el software.

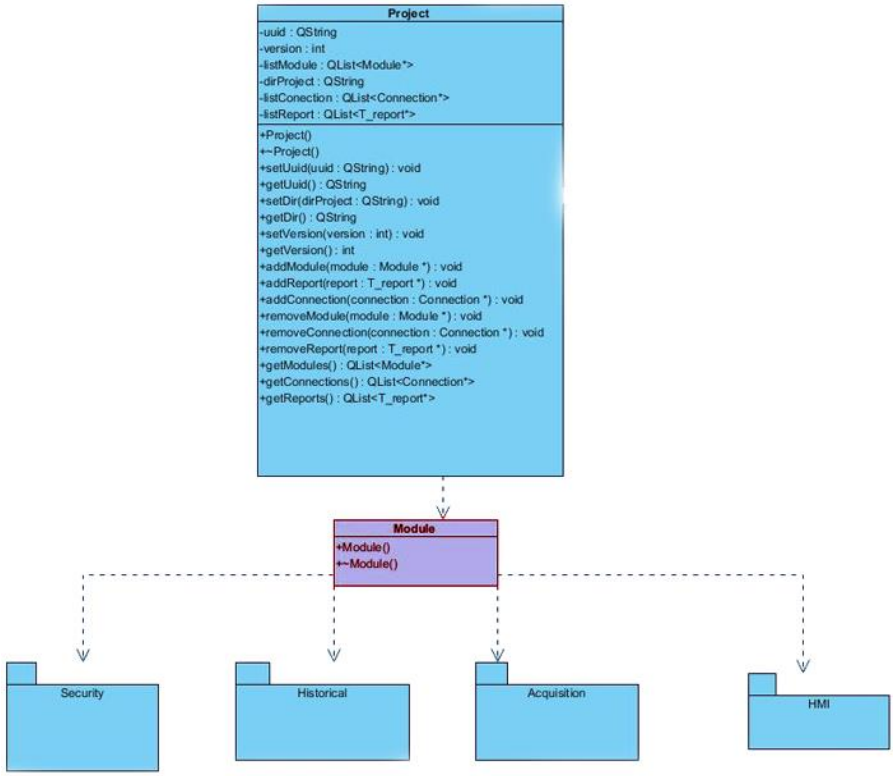


Figura 14: Ejemplo de patrón iterator en el diagrama del paquete Principal

- ***Observer*:** Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos. Está presente en el paquete *Acquisition* con el método ***updateMapMeasurement()*** cuya función es actualizar el mapa de mediciones cuando llegan cambios de configuración en caliente.

2.6. Diagrama de Paquetes

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

Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Los paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerid, *[62]*.

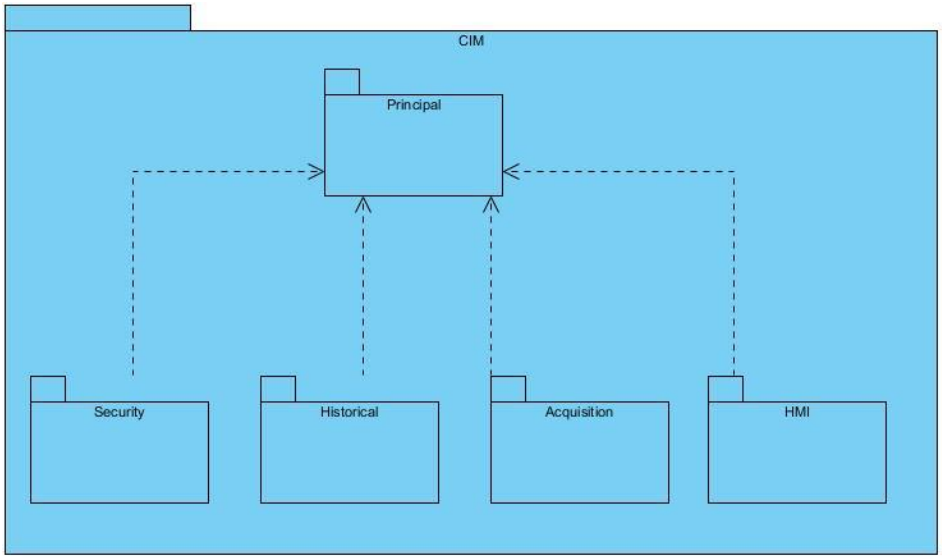


Figura 15: Diagrama de Paquetes General de la propuesta de solución

2.7. Diagrama de Clases

En ingeniería de *software*, un Diagrama de Clases (DC) es una forma de representación de los principales elementos que componen el diseño lógico de un sistema, describiendo la estructura de este y representando información modelable como atributos, operaciones y relaciones entre los objetos. UML define un conjunto de elementos que enriquecen la información brindada por estos diagramas como pueden ser la visibilidad (*public, private, protected*), la multiplicidad o frecuencia con que se instancian o asocian entidades u objetos dentro del modelo, así como el tipo de dato que reciben y retornan las operaciones.

En la propuesta de solución documentada para esta investigación se modelaron cinco DC subdivididos por paquetes para mejorar la comprensión de los mismos. En el paquete *Security* se relacionaron las clases correspondientes a la gestión de usuarios, grupos y perfiles, así como los permisos asociados a estas entidades. Mientras que en el paquete *Acquisition* se manejan elementos (clases) como los puntos, las conexiones, canales y subcanales, dispositivos y alarmas, así como la relación que existe entre estas entidades. Finalmente, en los paquetes *HMI* y *Historical* se hace la gestión de información referente a los registros de datos históricos y de consola; destacando que en cada caso solo se representan las clases principales del funcionamiento del sistema y la información que tribute a la solución del problema de investigación planteado

en el marco teórico. A continuación, se muestran los DC elaborados para los paquetes *Security*, *Acquisition* y *Principal*:

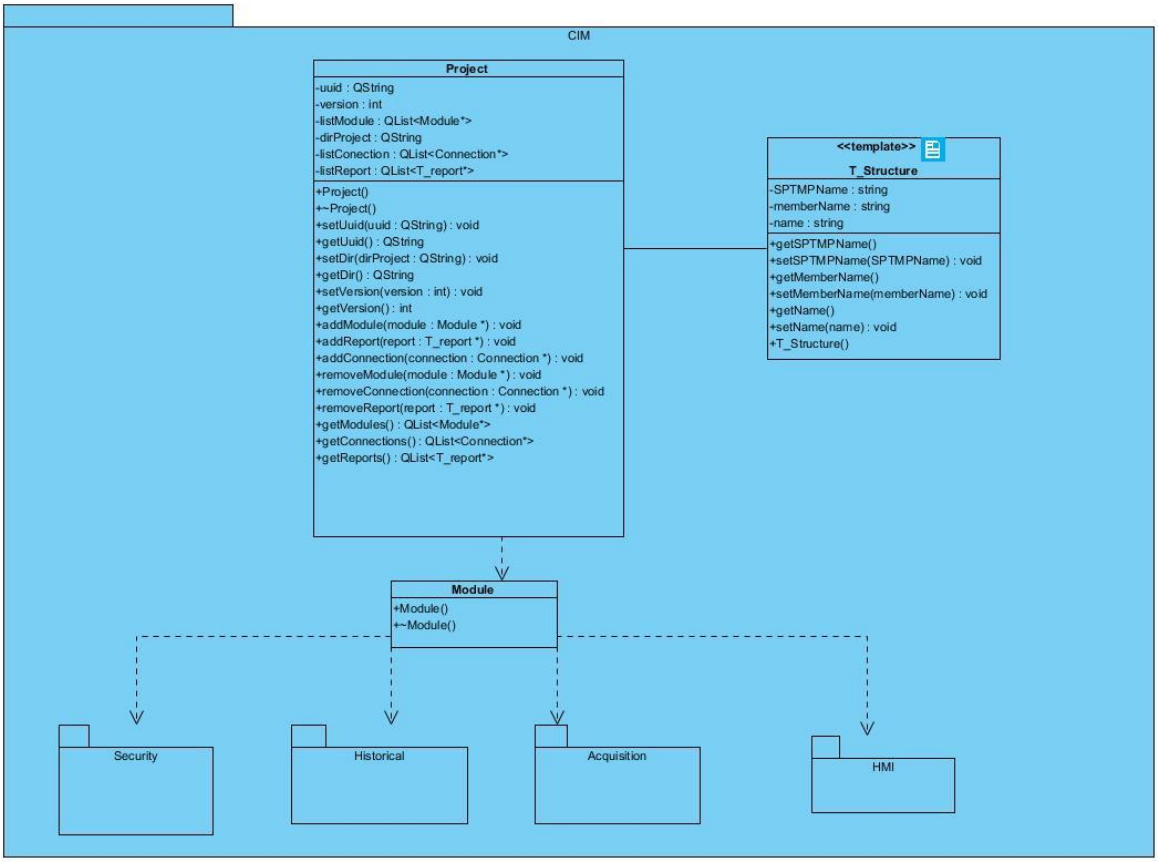


Figura 16: Diagrama de Clases del sistema, módulo Principal.

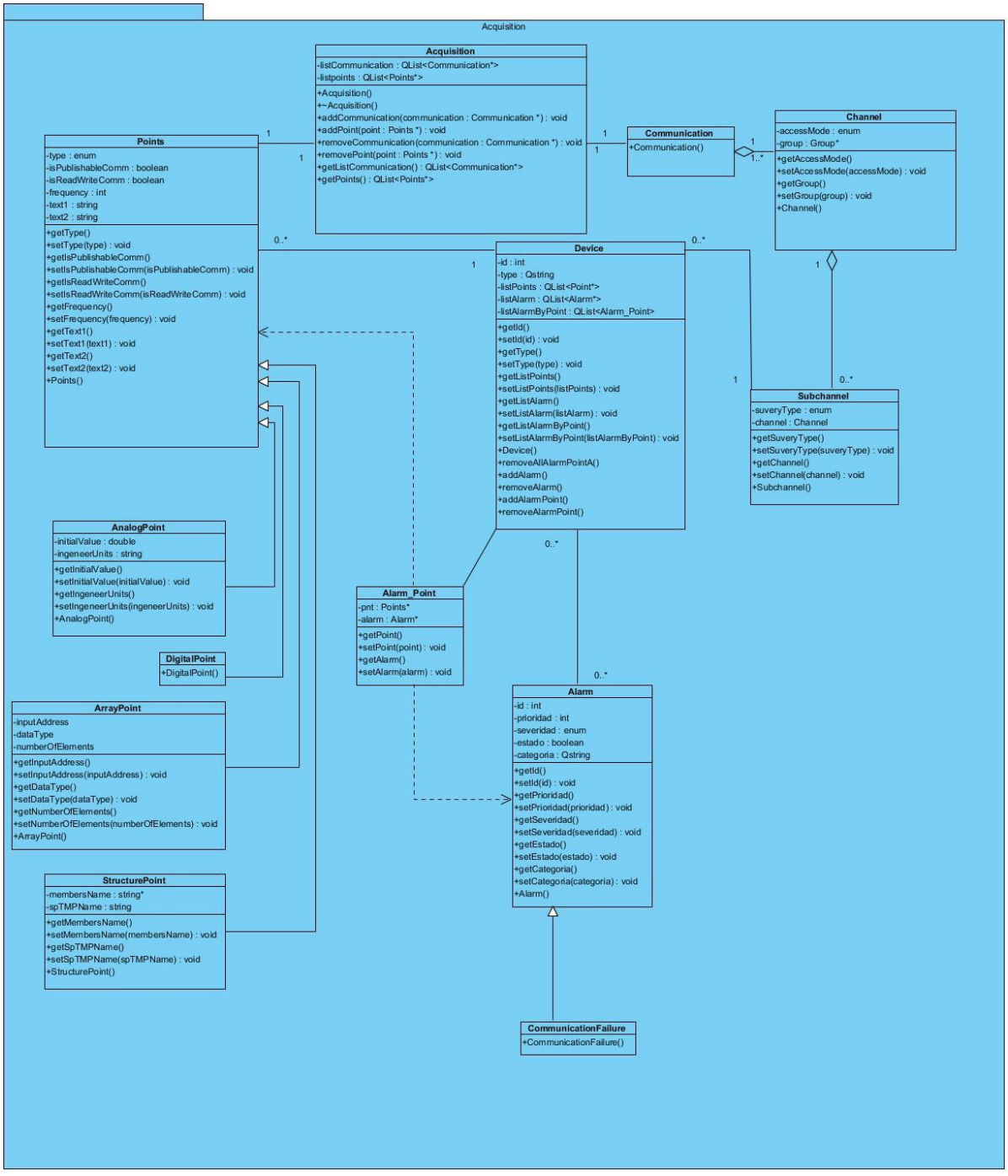


Figura 17: Diagrama de Clases del sistema, módulo Acquisition.

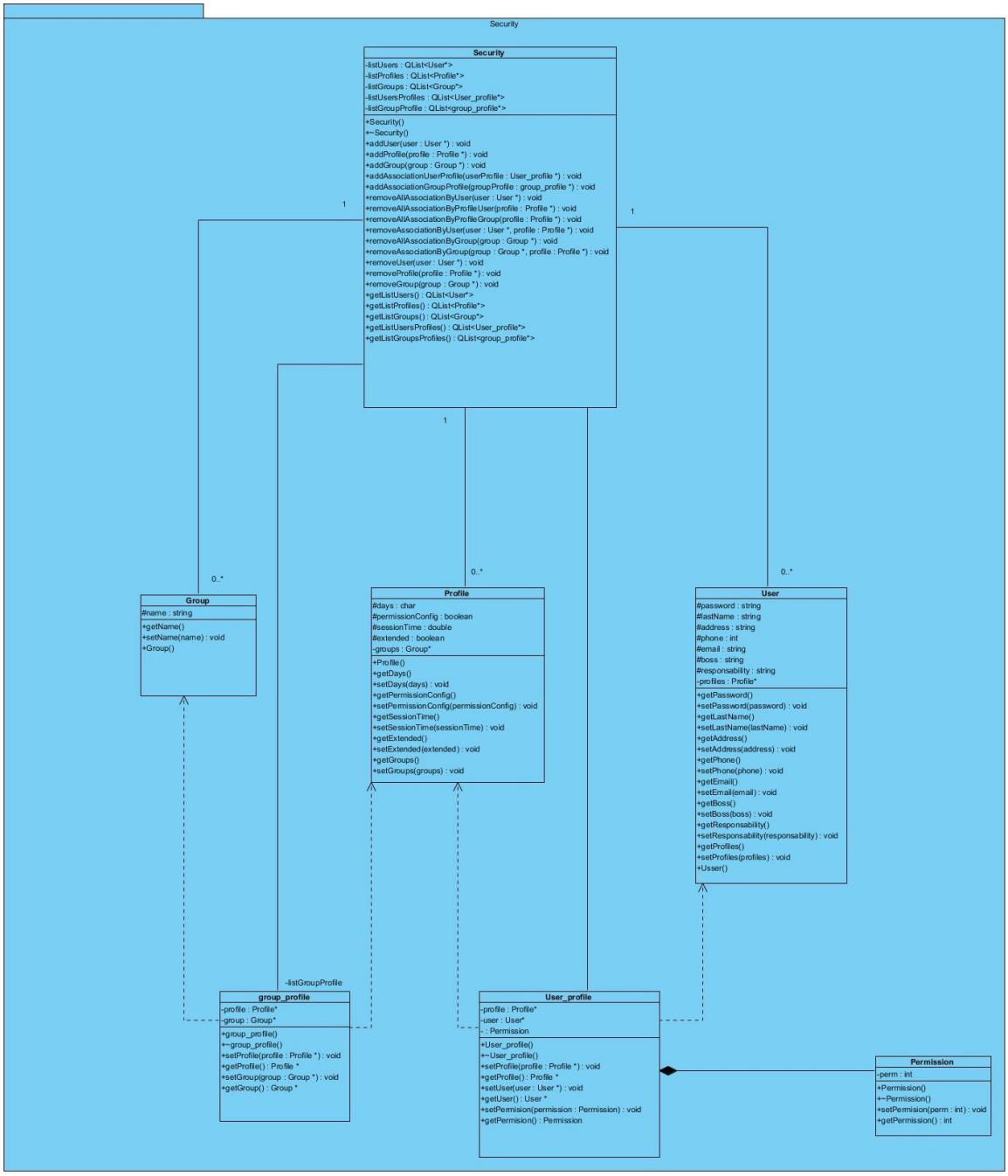


Figura 18: Diagrama de Clases del sistema, módulo Security

2.8. Diagrama de componentes

Un diagrama de componentes representa cómo un sistema divide los elementos que componen el *software* y muestra las dependencias entre ellos. Estos elementos, comúnmente conocidos como componentes físicos, puede ser archivos, bibliotecas, módulos, ejecutables, paquetes, archivos de clases o servidores.

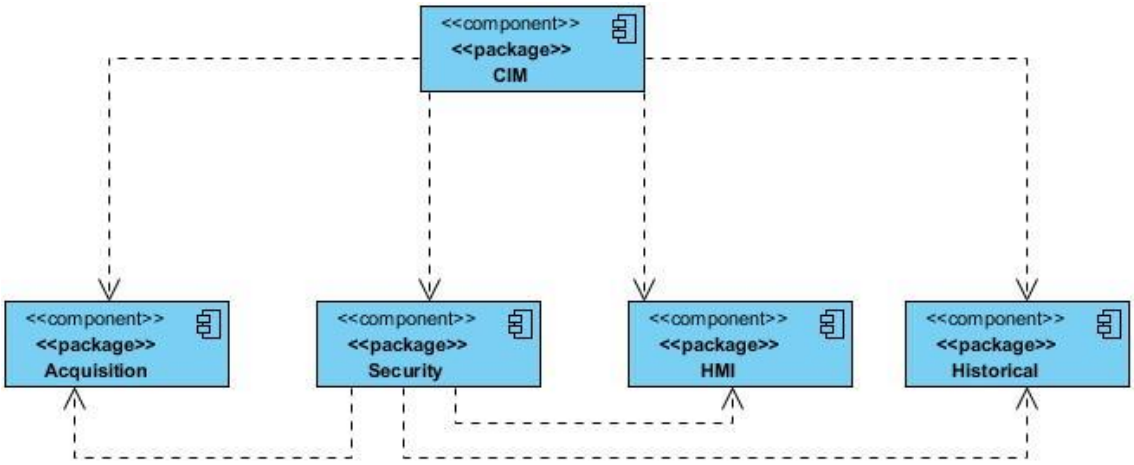


Figura 19: Diagrama de componentes del modelo de datos CIM

A continuación, se describen los componentes presentados en la figura 10 correspondiente al modelo de datos CIM de la propuesta de solución:

- ***CIM:*** paquete principal del modelo de datos que contiene las clases principales como *Project*.
- ***Acquisition:*** paquete de adquisición de datos del sistema SCADA que contiene las clases correspondientes las alarmas, puntos, dispositivos, entre otros.
- ***Security:*** paquete de seguridad que contiene las clases para la gestión de usuarios, perfiles, grupos y permisos (GOP).
- ***HMI:*** paquete correspondiente a la interfaz gráfica que gestiona el modelo. Agrupa las clases para la gestión de información asociada a este módulo como es console.
- ***Historical:*** paquete que contiene las clases para el registro histórico de los sucesos del *software* y la actividad relacionada con el modelo.

2.9. Historia de usuario

Una historia de usuario es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Las historias de usuario son utilizadas en las metodologías de desarrollo para la especificación de requisitos. Cada historia de usuario debe ser limitada y permitir responder rápidamente a los requisitos cambiantes *[47]*. La *Tabla 1:* Historia de Usuario muestra la historia de usuario número 1 que responde al RF 1, las restantes se comportan de igual forma desglosando las funciones de los paquetes para lograr una mejor comprensión.

Tabla 1: Historia de Usuario HU-1 para la gestión de los datos del paquete Security

Historia de usuario	
Identificador: HU-1	**Nombre de historia:** gestionar usuario del paquete *Security*
Referencia:HU-3	**Iteración asignada:** 1
Prioridad en negocio: alta	**Puntos estimados:** 1
Nivel de complejidad: alta	**Puntos reales:** 1
Descripción: el modelo debe permitir la inserción, eliminación y modificación de los usuarios del paquete *Security*.	

Tabla 2: Historia de Usuario HU-1.2 para la gestión de los datos del paquete Security

Historia de usuario	
Identificador: HU-1.2	**Nombre de historia:** gestionar perfil del paquete *Security*
Referencia: HU-1	**Iteración asignada:** 1
Prioridad en negocio: alta	**Puntos estimados:** 1
Nivel de complejidad: alta	**Puntos reales:** 1
Descripción: el modelo debe permitir la inserción, eliminación y modificación de los perfiles del paquete *Security*	

Tabla 3: Historia de Usuario HU-1.3 para la gestión de los datos del paquete Security

Historia de usuario	
Identificador: HU-1.3	**Nombre de historia:** gestionar grupo del paquete *Security*
Referencia: HU-1	**Iteración asignada:** 1
Prioridad en negocio: alta	**Puntos estimados:** 1
Nivel de complejidad: alta	**Puntos reales:** 1
Descripción: el modelo debe permitir la inserción, eliminación y modificación de los grupos del paquete *Security*	

Tabla 4: Historia de Usuario HU-1.4 para la gestión de los datos del paquete Security

Historia de usuario	
Identificador: HU-1.4	**Nombre de historia:** gestionar asociaciones del paquete *Security*
Referencia: HU-1	**Iteración asignada:** 1
Prioridad en negocio: alta	**Puntos estimados:** 2
Nivel de complejidad: alta	**Puntos reales:** 1.8
Descripción: el modelo debe permitir el manejo de las relaciones entre los grupos, perfiles y usuarios de forma centralizada logrando la menor dependencia entre ellos, de forma que cuando se elimine alguno de ellos no sean afectadas otras entidades y solo sea eliminada la asociación existente.	

Tabla 5: Historia de Usuario HU-2 para la gestión de los datos del paquete Acquisition

Historia de usuario	
Identificador: HU-2	**Nombre de historia:** gestionar canales del paquete *Acquisition*
Referencia: HU-3	**Iteración asignada:** 2
Prioridad en negocio: alta	**Puntos estimados:** 1
Nivel de complejidad: alta	**Puntos reales:** 1
Descripción: el modelo debe permitir la inserción, eliminación y modificación de los canales del paquete *Acquisition*	

Tabla 6: Historia de Usuario HU-2.1 para la gestión de los datos del paquete Acquisition

Historia de usuario	
Identificador: HU-2.1	**Nombre de historia:** gestionar subcanales del paquete *Acquisition*
Referencia: HU-2	**Iteración asignada:** 2
Prioridad en negocio: alta	**Puntos estimados:** 1
Nivel de complejidad: alta	**Puntos reales:** 1
Descripción: el modelo debe permitir la inserción, eliminación y modificación de los subcanales del paquete *Acquisition*	

Tabla 7: Historia de Usuario HU-2.2 para la gestión de los datos del paquete Acquisition

Historia de usuario	
Identificador: HU-2.2	**Nombre de historia:** gestionar dispositivos del paquete *Acquisition*
Referencia: HU-2	**Iteración Asignada:** 2
Prioridad en negocio: alta	**Puntos estimados:** 1
Nivel de complejidad: alta	**Puntos reales:** 1
Descripción: el modelo debe permitir la inserción, eliminación y modificación de los dispositivos del paquete *Acquisition*	

Tabla 8: Historia de Usuario HU-2.3 para la gestión de los datos del paquete Acquisition

Historia de usuario	
Identificador: HU-2.3	**Nombre de historia:** gestionar puntos del paquete *Acquisition*
Referencia: HU-2	**Iteración asignada:** 2
Prioridad en negocio: alta	**Puntos estimados:** 2
Nivel de complejidad: alta	**Puntos reales:** 1.5
Descripción: el modelo debe permitir la inserción, eliminación y modificación de los puntos del paquete *Acquisition*. Los puntos pueden ser analógicos (*analogPoint*), digitales (*digitalPoint*), estructurales (*structurePoint*) y de colección (*arrayPoint*).	

Tabla 9: Historia de Usuario HU-2.4 para la gestión de los datos del paquete Acquisition

Historia de usuario	
Identificador: HU-2.4	**Nombre de hHistoria:** gestionar alarmas del paquete *Acquisitiona*
Referencia: HU-2	**Iteración asignada:** 2
Prioridad en negocio: alta	**Puntos estimados:** 2
Nivel de complejidad: alta	**Puntos reales:** 1.8
Descripción: el modelo debe permitir la inserción, eliminación y modificación de las alarmas del paquete *Acquisition*. Las alarmas pueden pertenecer a los dispositivos y a los puntos, con asociaciones entre ellos.	

Tabla 10: Historia de Usuario HU-2.5 para la gestión de los datos del paquete Acquisition

Historia de usuario	
Identificador: HU-2.5	**Nombre de historia:** gestionar asociaciones del paquete *Acquisition*
Referencia: HU-2	**Iteración asignada:** 2
Prioridad en negocio: alta	**Puntos estimados:** 2
Nivel de nomplejidad: alta	**Puntos reales:** 1.5
Descripción: el modelo debe permitir el manejo de las relaciones entre los canales y subcanales, los subcanales y dispositivos, los dispositivos y los puntos, los puntos y las alarmas y los dispositivos y las alarmas de forma centralizada logrando la menor dependencia entre ellos, de forma que cuando se elimine alguno de ellos no sean afectadas otras entidades y solo sea eliminada la asociación existente.	

Tabla 11: Historia de Usuario HU-3 para la gestión de los datos del paquete Principal

Historia de usuario	
Identificador: HU-3	**Nombre de historia:** gestionar los datos del paquete Principal
Referencia: -	**Iteración asignada:** 2
Prioridad en negocio: alta	**Puntos estimados:** 1
Nivel de complejidad: alta	**Puntos reales:** 0.7
Descripción: el modelo debe permitir la gestión de proyectos que contiene los módulos principales (*Acquisition, Security, HMI, Historical*) así como la gestión de estos módulos descritos en el diagrama de clases *Principal*.	

2.10. Plan de entregas

El plan o cronograma de entregas establece qué HU son agrupadas para conformar una entrega y el orden de las mismas. Este cronograma es el resultado de una reunión entre todos los actores del proyecto (cliente, desarrolladores, gerentes, entre otros). El cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores *[48]*.

A partir del análisis realizado inicialmente y en correspondencia con el mismo se realiza el plan de entregas, que se muestra en la tabla 2, en el cual se proponen dos versiones no funcionales, una funcional y una última entrega de iteraciones del producto el 23 de marzo, para dar paso a la fase de producción.

Tabla12: Plan de entregas

Historias de usuario	1ra Iteración 11 de enero del	2da Iteración 23 de marzo	3ra Iteración 20 de abril del	4ta Iteración 15 de mayo del

	2016	del 2016	2014	2016
HU-1	V1.0			
HU-2				
HU-3		V1.1		
HU-4				
HU-5				
HU-1			V1.2	
HU-2				
HU-3				
HU-4				
HU-5				
HU-1				V1.3
HU-2				
HU-3				
HU-4				
HU-5				

Conclusiones parciales

En este capítulo se comenzó a profundizar en el desarrollo de la propuesta de solución, partiendo del análisis de los requisitos funcionales y no funcionales, así como la elaboración del modelo de dominio a partir de los principales conceptos asociados obteniéndose una lista de las funcionalidades que debe tener el modelo CIM, las mismas fueron representadas mediante historias de usuarios y luego fueron descritas. Se confeccionó el diagrama de paquetes para representar las dependencias entre los paquetes que componen el modelo, así como los diagramas de clase y el diagrama de componente. A partir de estos elementos se puede comenzar a construir el sistema, cumpliendo con todos los requerimientos y las funcionalidades que se consideraron en este capítulo.

CAPÍTULO 3: "PRUEBAS Y VALIDACIÓN DE LA SOLUCIÓN"

3. Introducción

En este capítulo se realiza una descripción de la última fase de la metodología aplicada referentes al correcto funcionamiento del producto, además se realizan las pruebas de caja blanca y rendimiento para validar los requerimientos del cliente.

3.1. Pruebas aplicadas al modelo CIM de la propuesta

Las pruebas de *software* son un elemento crítico para la garantía de la calidad del *software* y representan una revisión final de las especificaciones, del diseño y de la codificación. El objetivo fundamental de las pruebas es descubrir diferentes clases de errores con la menor cantidad de tiempo y de esfuerzo. Aunque las pruebas no pueden asegurar la ausencia de defectos; sí pueden demostrar que existen defectos en el *software* *[49]*.

Estas actividades se planean con anticipación y se realizan de manera sistemática. Cuando se aplican pruebas a un *software* es necesario tener en cuenta el objetivo que se persigue, debido a quelas pruebas son agrupadas por niveles que se encuentran en distintas etapas del proceso de desarrollo *[50]*.

En la metodología AUP-UCI se definen como disciplina tres tipos de pruebas (ver *Figura 20*), para las que se realizan los diferentes artefactos que se relacionan en este documento y se muestran los resultados obtenidos en cada uno de ellos. Además, dando cumplimiento a los requisitos recogidos al cliente, se realizaron solamente las pruebas internas (caja blanca) y las pruebas de rendimiento.

Para la propuesta de solución presentada se prescinde de las pruebas de liberación que estipula la metodología. Estas pruebas solo pueden ser realizadas por una entidad certificadora de calidad externa y en condiciones de términos de tiempo pactados excedería la planificación realizada.

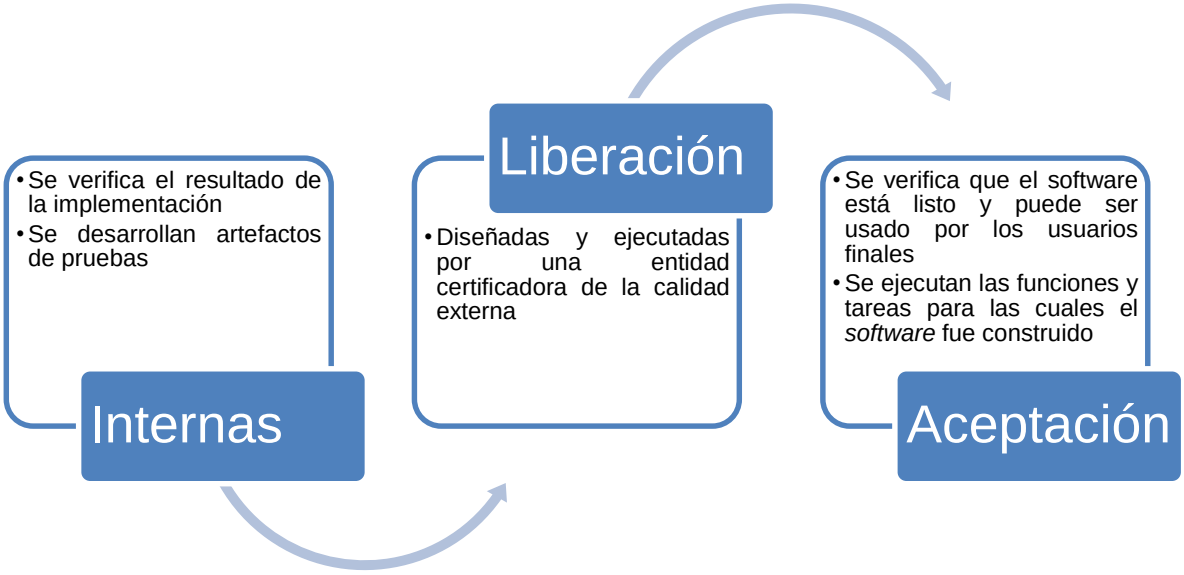


Figura 20: Tipos de pruebas descritas en la metodología AUP-UCI y sus principales características.

Cualquier producto de ingeniería se puede probar de tres formas:

- **Pruebas de caja negra**: Realizar pruebas de forma que se compruebe que cada función es operativa.
- **Pruebas de caja blanca**: Desarrollar pruebas de forma que se asegure que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han probado de forma adecuada.
- **Pruebas automatizadas**: Aplicar pruebas con el uso de *software* especial para controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados. La automatización de pruebas permite incluir pruebas repetitivas y necesarias dentro de un proceso formal de pruebas ya existente o bien adicionar pruebas cuya ejecución manual resultaría difícil.

En la prueba de caja blanca se realiza un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos del programa, además de los bucles y condiciones, y examinado el estado del programa en varios puntos. Comprobar el 100% del código garantizaría un programa completamente sin errores, pero esto supone un estudio demasiado exhaustivo, que prolongaría excesivamente los planes de desarrollo del *software*, por lo que se hará un estudio de los caminos lógicos importantes.

3.1.1. Prueba de camino básico

El método del camino básico (propuesto por McCabe) permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.

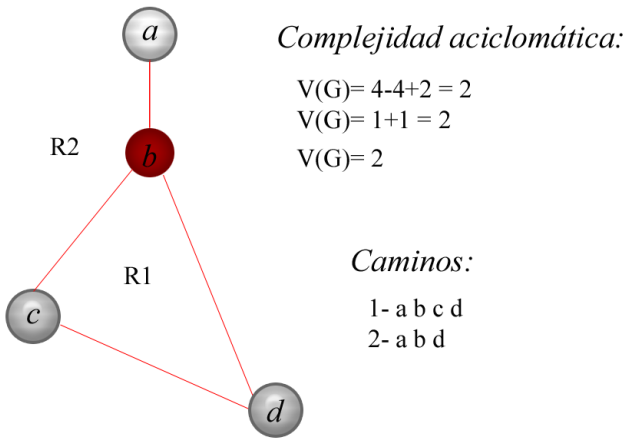
¿Cuáles son los pasos a seguir para aplicar una prueba de camino básico?

- Obtener el grafo de flujo, a partir del diseño o del código del módulo.
- Obtener la complejidad ciclomática del grafo de flujo.
- Definir el conjunto básico de caminos independientes.
- Determinar los casos de prueba que permitan la ejecución de cada uno de los caminos anteriores.
- Ejecutar cada caso de prueba y comprobar que los resultados son los esperados.

En la presente investigación se realizaron cuatro pruebas de camino básico sobre los métodos críticos o métodos que mejor reflejan el flujo de otros con similar funcionamiento. Como ejemplo se observa en la tabla siguiente el Diseño de Caso de Prueba (DCP) para el método ***addModule()*** con dos posibles escenarios, el resto de las pruebas se relacionan a continuación.

Tabla 13: DCP 1 aplicando técnica de camino básico

Numero: 1	**Método:** *addModule*	**Clase:** *Project*

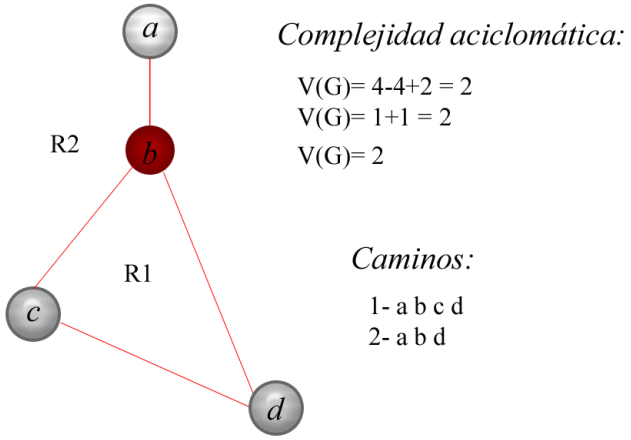


Escenario	Descripción	Valor	Respuesta esperada	Respuesta del sistema	Evaluación de prueba

EC.1: adicionar módulo al proyecto	No existe el módulo	V Acquisition2	Se adiciona el módulo al proyecto	Se adiciona el módulo al proyecto	satisfactoria
EC.2: adicionar módulo proyecto	Existe el módulo	V Acquisition2	No se adiciona el módulo al proyecto	No se adicionó el módulo al proyecto	satisfactoria

Tabla 14: DCP 2 aplicando técnica de camino básico

Numero: 2 **Método:** *removeModule* **Clase:** *Project*

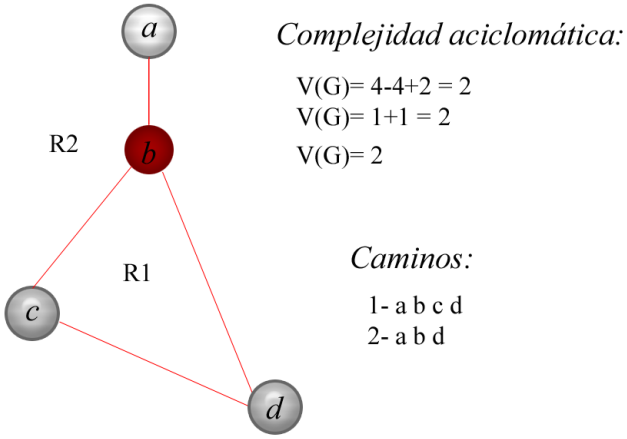


Escenario	Descripción	Valor	Respuesta esperada	Respuesta del sistema	Evaluación de prueba
EC.1: eliminar módulo del proyecto	No existe el módulo	F PruebaModuleFalse10	No se elimina el módulo y sale del procedimiento	No se elimina el módulo y sale del procedimiento	satisfactoria
EC.2: eliminar módulo del	Existe el módulo	V	Se elimina el módulo	Se elimina el módulo	satisfactoria

proyecto		Acquisition2			

Tabla 15: DCP 3aplicando técnica de camino básico

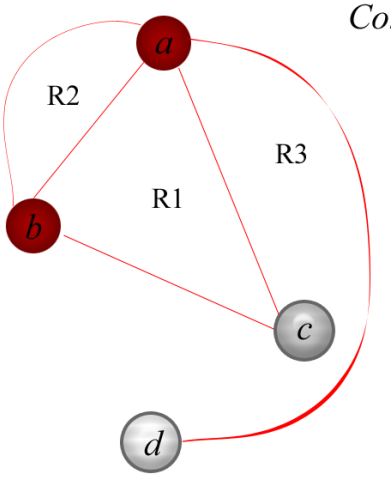
Numero: 3	**Método:** *addGroup*	**Clase:** *Security*



Escenario	Descripción	Valor	Respuesta esperada	Respuesta del sistema	Evaluación de prueba
EC.1: adicionar grupo de seguridad	No existe el grupo	V Grupo2	Se crea y adiciona el grupo a la lista de la clase *Security*	Se crea y adiciona el grupo a la lista de la clase *Security*	satisfactoria
EC.2: adicionar grupo de seguridad	Existe el grupo	V Grupo2	No se adiciona el grupo y sale del procedimiento	No se adiciona el grupo y sale del procedimiento	satisfactoria

Tabla 16: DCP 4 aplicando técnica de camino básico

Numero: 4	**Método:** *removeAllAssociationByProfileUser*	**Clase:** *Security*



Complejidad aciclomática:

V(G)= 5-4+2 = 3
V(G)= 2+1 = 3
V(G)= 3

Caminos:

1- a b c a d
2- a b a d
3- a d

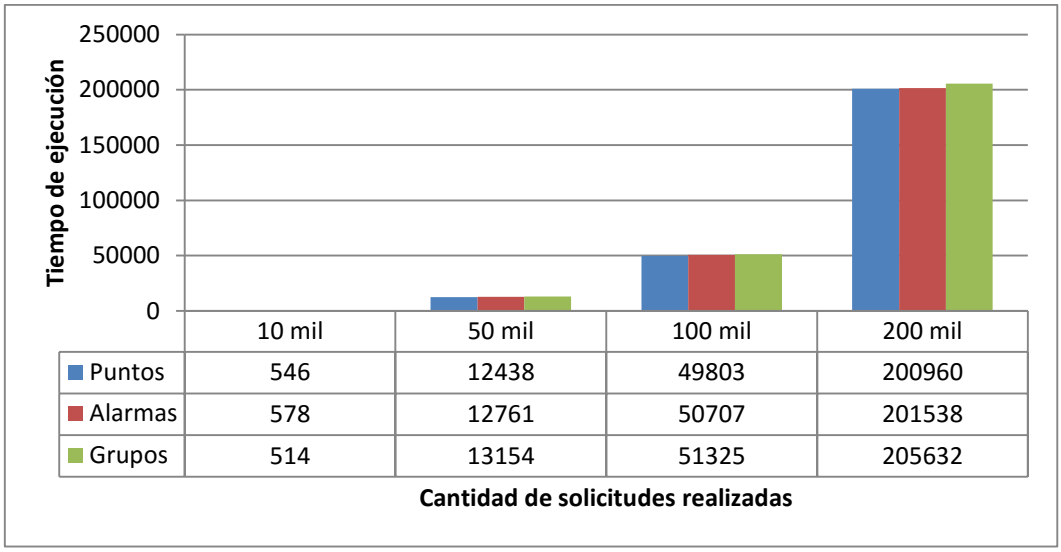
Escenario	Descripción	Valor	Respuesta esperada	Respuesta del sistema	Evaluación de prueba
EC.1: eliminar asociaciones de perfil y usuario dado un pefil	Existen asociaciones con el perfil introducido	V Admin10	Elimina todas las asociaciones existentes con el perfil introducido	Elimina todas las asociaciones existentes con el perfil introducido	satisfactoria
EC.2: eliminar asociaciones de perfil y usuario dado un perfil	No existen asociaciones con el perfil introducido	V Admin10	No se elimina ninguna asociación y se sale del procedimiento	No se elimina ninguna asociación y se sale del procedimiento	satisfactoria
EC.3: eliminar asociaciones de perfil y usuarios dado un perfil	No existen asociaciones en la lista	V AdminFalse	No se elimina ninguna asociación y se sale del procedimiento	No se elimina ninguna asociación y se sale del procedimiento	satisfactoria

3.1.2. Prueba de rendimiento

Las pruebas de rendimiento son aquellas que se le realizan a un sistema, bajo condiciones específicas, para determinar la rapidez con que responden a una o varias solicitudes por parte del usuario u otra entidad (clase/objeto/*software*). Generalmente son utilizadas para medir la calidad del *software* en cuanto a escalabilidad, fiabilidad y uso de los recursos garantizando mejorar el rendimiento, detectar errores de codificación como la fuga de memoria y

además se demuestra que el sistema cumple con los criterios de rendimiento establecidos por el usuario o el ambiente de ejecución.

Como parte de la fase de prueba del proceso de desarrollo de *software*, se aplicaron al modelo de solución propuesto las pruebas de rendimiento en cuanto a tiempo de respuesta de las solicitudes o instanciaciones de objetos (clases) como puntos y alarmas. Además, se valoró el consumo de memoria física como uno de los recursos críticos a tener en cuenta para garantizar un mejor funcionamiento. A continuación, se grafican los resultados arrojados por dichas pruebas:



	10 mil	50 mil	100 mil	200 mil
Puntos	546	12438	49803	200960
Alarmas	578	12761	50707	201538
Grupos	514	13154	51325	205632

Gráfico 1: Prueba de rendimiento (tiempo) para las clases Point, Alarm y Group

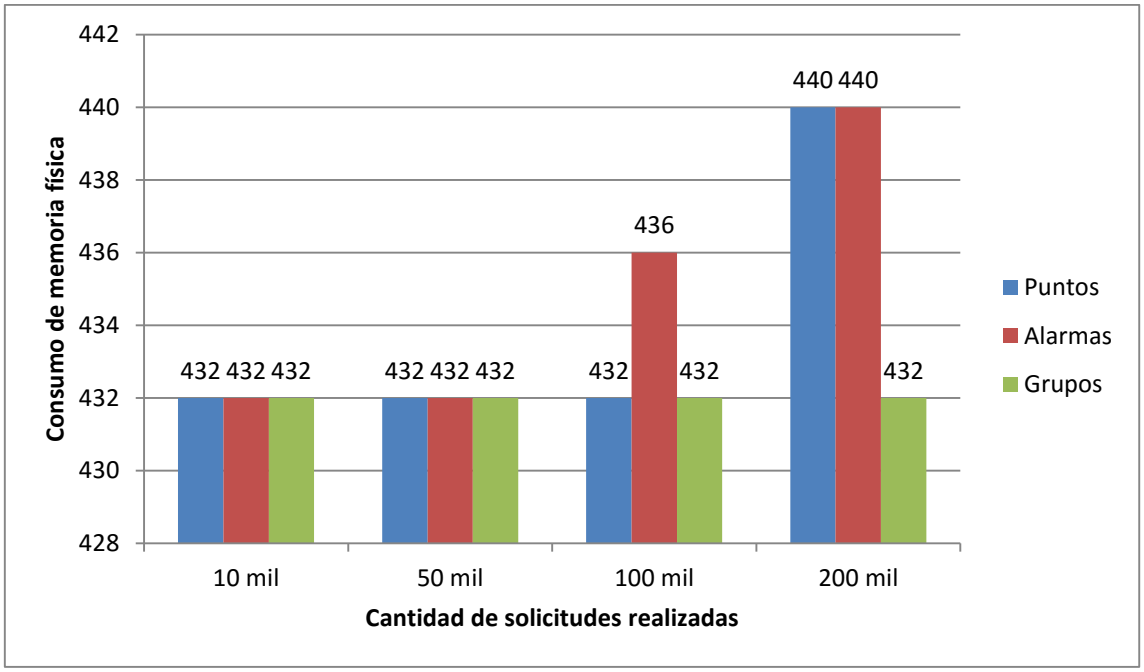


Gráfico 2: Prueba de rendimiento (memoria) para las clases Point, Alarm y Group

El entorno de las pruebas se basa en las herramientas y tecnologías descritas en el capítulo 1, mientras que el *hardware* que dio soporte presenta las siguientes características:

- Modelo: HP Pavilion v6 (laptop)
- Procesador: AMD Sempron 2.10 GHz
- Memoria RAM: 2.00 GB (1.75 GB utilizable)
- Sistema operativo: Windows 7 Ultimate, 32 bits

Las pruebas resultaron satisfactorias en su totalidad para la segunda iteración de los procedimientos evaluados, por lo que se puede concluir que el modelo presenta buena estructura de código y un desarrollo procedural eficaz. Las pruebas de rendimiento, por su parte, generaron estadísticas sobre el consumo de memoria física y el tiempo de ejecución del *software* consolidando los requerimientos obtenidos y dentro de los criterios establecidos. El tiempo promedio de ejecución en un rango de 10000 a 200000 solicitudes fue de 65936 ms, 66396 ms y 67656 ms para los puntos, las alarmas y los grupos respectivamente. Por otra parte, el consumo de memoria física promedio registrado para el mismo rango de solicitudes fue de 434 kb, 435 kb y 432 kb para puntos, alarmas grupos respectivamente. Para las solicitudes, con cifras inferiores a 1000, el modelo consume menos de 1 ms para todos los casos instanciados.

Conclusiones parciales

Se realizaron pruebas de caja blanca y rendimiento para validar la propuesta de solución. Las pruebas destinadas al código, como las de camino básico, fueron aplicadas a los métodos críticos que presentan mayor similitud con otros de igual función como son ***addUser()*** y ***removeAllAssociationProfileUser()***, entre otros. Estos resultados en conjunto dan una valoración del proceso de prueba y calidad realizado al modelo de datos propuesto en la solución de la investigación, proceso que representa una fase esencial en todo ciclo de vida de un *software*.

CONCLUSIONES GENERALES

Para la implementación y correcto funcionamiento de un sistema SCADA, el modelo de datos CIM representa uno de los pilares para *software* de esta categoría. Constituye la base del manejo de la información y la representación estructurada de los datos tributando a la integridad y estabilidad funcional del mismo. Además, garantiza, con el uso de estilos, patrones y modelos que se logre obtener un sistema escalable, con altas facilidades de mantenimiento y un buen rendimiento.

Durante el desarrollo de la investigación se cumplieron los objetivos propuestos diseñándose e implementándose un modelo de datos CIM que permite la organización de la información del sistema SCADA SAINUX, concluyéndose que:

- Con el desarrollo del modelo capaz de organizar y estructurar la información del sistema SCADA SAINUX se eliminó la información redundante y el alto acoplamiento de las clases permitiendo un mejor manejo de las asociaciones.
- Mediante las iteraciones de pruebas realizadas al sistema se logró resolver las no conformidades detectadas que dan solución a los requisitos de rendimiento especificados, posibilitando además la realización de operaciones de actualización, así como la correcta liberación de la memoria.
- Con el resultado del presente trabajo se da cumplimiento a los requisitos planteados por el proyecto SCADA SAINUX correspondiente a la organización de información mediante un modelo de datos común.

RECOMENDACIONES

Con el objetivo de mejorar el modelo de datos, así como de implementar otras funcionalidades que optimicen el rendimiento se plantean las siguientes recomendaciones:

- Agregar al modelo de datos los objetos (clases) que facilitan el manejo coherente de información como el paquete *Core* y *MEAs*.
- Agregar al modelo de datos las operaciones (métodos) auxiliares del modelo anterior que brindan funcionalidades específicas como *updateMapMeasurement().*
- Integrar el modelo de datos CIM de la propuesta de solución al sistema SCADA SAINUX.

REFERENCIAS BIBLIOGRÁFICAS

1. García Giménez, Javier. Calibración, Control y Diseño SCADA de un robot paralelo neumático con el autómata S7-300. SCADA, una primera idea. 2008 [citado 26/12/2015]; Disponible en: http://repositorio.bib.upct.es/dspace/bitstream/10317/570/1/pfc2683.pdf.
2. Martínez Gonzales, Yadiel y del Rosario Lalcebo, Yanelys. Subsistema de comunicaciones para el SCADA SAINUX. Tesis de diploma inédita. Universidad de las Ciencias Informáticas, 2012.
3. GÓMEZ, Oscar Tinoco, LÓPEZ, Pedro Pablo Rosales and BACALLA, Julio Salas, 2010, Criterios de selección de metodologías de desarrollo de *software*. Industrial Data. 2010. Vol. 13, no. 2, p. 70–74.
4. SÁNCHEZ ELÍAS, L. Mecanismo de disponibilidad para el Configurador SCADA UX. 2011.
5. Cabrera Robles, Henry Marcelo. Mecanismo de configuración en caliente para la interfaz hombre máquina del sistema SCADA UX. Tesis de diploma inédita. Universidad de las Ciencias Informáticas, 2013.
6. Herrera Vázquez, Moisés. Introducción a la Arquitectura del Guardián del ALBA. 2008.
7. HONEYWELL. PlantScape Disponible en: http://www.honeywellsp.com/hw_productos_servicios/hw_industrial/Hw_Control_Automatizacion_Industrial.htm.
8. Sistemas HMI. Disponible en: http://www.generatecnologias.es/sistemas_hmi.html.
9. Castillo Chacón, Carlos Manuel y Acevedo Medina, Jose Manuel. Procesador de estado de red para adecuaciones del SCADA-GALBA al sector eléctrico. Tesis de diploma inédita. Universidad de las Ciencias Informáticas, 2015.
10. Lozano, Carlos de Castro. Introducción SCADA.
11. InfoPLC. Top 5 de aplicaciones SCADA para iPhone y iPad.
12. Aragón Cáceres, José y Llanes Jiménez, A. Beatriz. Servicio de Integración con Terceros para el acceso a variables del sistema SCADA Guardián del ALBA. 2009 [citado 27/12/2015] Disponible en: http://bibliodoc.uci.cu/TD/TD_2554_09.pdf

13. “IEC 61850, El Nuevo Estándar en Automatización de Subestaciones”.

14. IEC_61970-301, “Energy management system application program interface (EMS-API) – Common information model (CIM) base,” 2007.

15. IEC_61970-552-4, “Energy management system application program interface (EMS-API) – CIM XML model Exchage Format,” 2005.

16. IEC_61970-1, “Energy management system application program interface (EMS-API) –Guidelines and general requirements,” 2007.

17. IEC_61968-1, “Application integration at electric utilities – Ssystem interfaces for distribution management – Interface architecture and general requirements,” 2003.

18. IEC_61970-402, “Energy management system application program interface (EMS-API) – Component interface specification (CIS) – Common services,” 2006.

19. IEC_61968-1, “Application integration at electric utilities – System interfaces for distribution management – Interface for network operations,” 2004.

20. Santodomingo Berry, Rafael. Introducción al modelo CIM de los sistemas de energía eléctrica. Canales de mecánica y electricidad. [en línea]. Septiembre-octubre 2009. [Consultada el: 27/12/2015].

21. LojanBermeo, Edgar Fernando e Iñiguez Quesada, Daniel Adrián. Diseño de un sistema HMI/SCADA para una planta de clasificación con Visión Artificial. 1ra ed. Facultad de Ciencia y Tecnología de la Universidad del Azuay: EGRIET, 2008.

22. OoCities. Introducción a los Sistemas SCADA. [citado 2016 Enero, 12]; Disponible en: http://www.oocities.org/gabrielordonez_ve/SISTEMAS_SCADA.htm

23. EcuRed. Sistema SCADA. [citado 2016 Enero, 12]; Disponible en: http://www.ecured.cu/Sistema_SCADA

24. CUEVAS, Ignacio Aedo, 2005, Ingeniería De La Web Y Patrones De Diseño. [online]. 2005. [Citado 2016 enero, 12]. Disponible en: http://dialnet.unirioja.es/servlet/libro?codigo=323311

25. Pujol Mendez, Naivy. Sistema de Gestión para el Centro Internacional de Postgrado. Tesis de diploma inédita. Universidad de las Ciencias Informáticas, 2014.

26. ALMEIRA, Adriana Sandra and PEREZ, Vanina, 2007, Arquitectura de Software: Estilos y Patrones [online]. Tesina. Argentina: Universidad Nacional De La Patagonia San Juan Bosco. Available from: http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_1/Arquitectura_de_Software/Tesina_Arquitectura_de_Soft.pdf

27. GAMMA, Erich, HELM, Richard, JOHNSON, Ralph and VLISSIDES, John, 2008, GoF Design Patterns with examples using Java and UML2 [online]. Autoedición. [Accessed 10 November 2013]. ISBN 0-201-63361-2. Available from: http://www.etnassoft.com/biblioteca/gof-design-patterns/

28. GUERRERO, Carlos A., SUÁREZ, Johanna M. and GUTIÉRREZ, Luz E., 2013, Patrones de Diseño GOF (TheGang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web. Información tecnológica. January 2013. Vol. 24, no. 3, p. 103–114. DOI 10.4067/S0718-07642013000300012.

29. HOLZNER, Steve, 2006, Design Patterns for Dummies [online]. Wiley Publishing, Inc. ISBN 0-471-79854-5. Available from: www.wiley.com

30. EcuRed. Patrón de diseño de *software*. [citado 2016 Enero, 12]; Disponible en: http://www.ecured.cu/Patrones_de_dise%C3%B1o_de_software

31. J. L. Cendejas Valdez, “Implementación del Modelo Integral Colaborativa (MDSIC) como fuente de innovación para el desarrollo ágil de *software* en las empresas de la zona centro occidente en México,” Tesis doctoral Inédita, Universidad Popular Autónoma del Estado de Puebla, 2014.

32. Someerville, I. (2005). Ingeniería del *software* (séptima ed.). (A. B. María Isabel Alfonso Galipienso, Trad.) Madrid, Madrid, España: Pearson.

33. Gacitúa Bustos, Ricardo A, “Métodos de desarrollo de *software*: El desafío pendiente de la estandarización,” 2003.

34. Silva, Darío Andrés y Mercerat, Bárbara, “Construyendo aplicaciones web con una metodología de diseño orientada a objetos,” 2001.Disponible en: www.lifia.info.unlp.edu.ar/papers/2001/Silva2001.pdf

35. Choque Aspiazu, Guillermo (2001). Ingeniería de requerimientos. Disponible en: http://www.espe.edu.ec/portal/files/sitio_congreso_2011/papers/C2.pdf

36. Pressman, R. Ingeniería de *software*. Un enfoque práctico. España, McGraw. Hill, 2005. 585 p.

37. Boeras Velázquez, Mairelys, “Aplicando el método de Boehm y Turner”. Serie Científica de la Universidad de las Ciencias Informáticas. No 6, vol. 5, 2012.

38. “Metodologías de desarrollo de *Software* - EcuRed.” [Online]. Available: http://www.ecured.cu/Metodologias_de_desarrollo_de_Software. [Accessed: 16-Jan-2016].

39. “MetodologíaCrystalIngeniería de *Software*.” [Online]. Available: http://ingenieriadesoftware.mex.tl/59189_Metodologia-Crystal.html. [Accessed: 16-Jan-2016].

40. “Essential Unified Process - EcuRed.” [Online]. Available: http://www.ecured.cu/index.php/Essential_Unified_Process. [Accessed: 16-Jan-2016].

41. “Agile Unified Process - EcuRed.” [Online]. Available: http://www.ecured.cu/Agile_Unified_Process. [Accessed: 16-Jan-2016].

42. RodriguezDonatien, Ariagna. Descripción de la Metodología de Desarrollo de *Software* Agile UnifiedProcess (AUP). Tesis de diploma inédita. Universidad de la Habana, 2011.

43. H. Hernández Feriñas, “Versión Miranda R2 del módulo de adquisición del SCADA Guardián del Alba,” Tesis de diploma inédita, Universidad de las Ciencias Informáticas, 2013.

44. SeharaDriggs, Yosell Luis. Implementación de un modelo para la configuración de un sistema SCADA. Ciudad de la Habana: s.n., 2008.

45. MEJORA, PROGRAMA DE. Metodología de Desarrollo para la Actividad Productiva de la UCI. [En línea] [Citado el: 2 de diciembre de 2015.]

46. Beck, K.Extreme Programming Explained. Embrace Change. s.l.: Pearson Education, 1999.

47. Mike Cohn, "User Stories Applied", 2004, Addison Wesley, ISBN 0-321-20568-5

48. JOSKOWICZ, José, 2008, Reglas y prácticas en Extreme Programming. Universidad de Vigo. España [online]. 2008. [Accessed 14 February 2014]. Available from: http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf

49. PRESSMAN, R. S. Ingeniería de *Software*, un enfoque práctico. Sexta edición ed.

50. LEYVA ABRAHANTES, D. Diseño e implementación de un módulo para la realización depresentaciones web reusables sobre Moodle. 2011.

51. EcuRed. Sistema de Supervisión y Control de Procesos EROS. [citado 2016 Enero, 12]; Disponible en: http://www.ecured.cu/Sistema_de_Supervisi%C3%B3n_y_Control_de_Procesos_EROS