Universidad de las Ciencias Informáticas

Facultad 5

Mecanismo de serialización para la información gestionada en el editor SCADA SAINUX.

Trabajo de Diploma para optar por el Título de
Ingeniero en Ciencias Informáticas.

Autor: Alvaro Denis Acosta Quesada.

Tutor: Ing. Ridel Oscar García Mora.

Ing. Leiser Fernández Gallo.

Co-Tutor: Ing. Luis Andrés Valido Fajardo.

La Habana, junio de 2016

"Año 57 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2016.

Autor:

_____

Alvaro Denis Acosta Quesada

Tutores:

_____ _____

Ing. Ridel Oscar García Mora Ing. Leiser Fernández Gallo

Co-Tutor:

_____

Ing. Luis Andrés Valido Fajardo

DATOS DE CONTACTO

Tutores:

Ing. Ridel Oscar García Mora: Arquitecto general del SCADA SAINUX con varios años de experiencia como desarrollador en el CEDIN. e-mail: rmora@uci.cu

Ing. Leiser Fernández Gallo: Ingeniero en Ciencias Informáticas desde 2015. Recién Graduado en Adiestramiento en la Dirección de Redes luego de 6 años como desarrollador del CEDIN. e-mail: leiserfg@uci.cu

Co-tutor

Ing. Luis Andrés Valido Fajardo: Desarrollador Sr. del SCADA SAINUX con varios años de experiencia en el CEDIN.

DEDICATORIA

A mis familiares y amigos, especialmente a mis tutores, mis padres y mi tío Ramón.

AGRADECIMIENTOS

A mis familiares y amigos, especialmente a mis tutores, mis padres y mi tío Ramón.

RESUMEN

El proceso de serialización y deserialización del proyecto SCADA actualmente presenta demora en las funcionalidades de cargar y guardar la información del proyecto de acuerdo al tamaño. Además, el tiempo de ejecución es lento teniendo en cuenta el tiempo promedio de ejecución de las acciones que se realizan, llegando a bloquearse completamente la aplicación. Para resolver el problema antes mencionado, la presente investigación tiene como objetivo desarrollar un mecanismo de serialización y deserialización que disminuya el tiempo de ejecución, para cargar y guardar la información gestionada en el ambiente de configuración de la Interfaz Hombre Maquina del sistema SCADA SAINUX. Se realizó un análisis a las soluciones similares para reconocer la existencia de bibliotecas diseñadas para la serialización y deserialización tanto binaria como de texto plano, permitiendo determinar *QdataStream* como solución de la propuesta. El desarrollo del mecanismo está guiado por la metodología AUP (variante Universidad de las Ciencias Informáticas) con el escenario #4. Se realizaron pruebas funcionales, del sistema y de aceptación, mediante la aplicación de la técnica de diseño de caja negra y el método de partición de equivalencia.

Palabras clave: bibliotecas, deserialización, *QdataStream*, SCADA, serialización

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Un sistema de supervisión, control y adquisición de datos SCADA[1] se refiere a un sistema central que monitorea y controla un sitio completo o un sistema que se extiende sobre una gran distancia (kilómetros / millas) (1). 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 en el SCADA SAINUX[3] el HMI consta de dos entornos: El entorno de configuración (EC), donde los mantenedores configuran la información específica del área que se desea supervisar y diseñan los despliegues, los cuales haciendo uso de los componentes gráficos permiten simular los procesos de campo; el entorno de visualización (EV) 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, monitorear los cambios de estado de las variables, gestionar alarmas, generar reportes, etc. Entre las funcionalidades que brinda el ambiente de configuración al mantenedor son las de abrir y guardar las configuraciones realizadas en el editor de forma local en la estación de trabajo. Esto es posible gracias a que toda la información gestionada referente al proceso o proyecto configurado en el editor es serializada en varios archivos en formato XML[4]. Esta serialización en formato XML fue factible para la cantidad de recursos a supervisar definidos para la primera versión del SCADA, pero el propio desarrollo de los módulos que componen al proyecto ha dado al traste, que ya no sea factible, el desarrollo de versiones posteriores implicó un aumento vertiginoso de la cantidad de recursos a supervisar por estos módulos. Todo lo anterior provoca que el mecanismo de serialización XML presente las siguientes deficiencias:

[1] SCADA, acrónimo del inglés, *Sypervisory Control and Data Adquisition*.

[2] HMI, acrónimo del inglés, *Human Machine Interface*.

[3] SAINUX, acrónimo de Sistema de Automatización Industrial para GNU/Linux.

[4] XML, acrónimo del inglés *Extensible Markup Language*.

- ✓ El tiempo de ejecución es lento teniendo en cuenta el tiempo promedio de ejecución de las acciones que se realizan, llegando a bloquearse completamente la aplicación a partir de un número de recursos configurados.
- ✓ La demora en el proceso de carga y salva de información provoca que cuando se está configurando un proyecto de gran tamaño, la usabilidad del entorno de edición disminuya pues el usuario debe esperar intervalos de tiempos injustificados para realizar una modificación por pequeña que sea.

De ahí que surja la siguiente interrogante, que constituye el **problema científico** del presente trabajo:

¿Cómo disminuir el tiempo de ejecución del mecanismo para guardar la configuración de manera local en el ambiente de edición del SCADA SAINUX?

Lo cual precisa como **objeto de estudio:** Los mecanismos empleados para guardar la configuración de manera local en ambientes de edición de SCADA.

Todo lo cual determina como **campo de acción**: Los mecanismos empleados para guardar la configuración de manera local en ambientes de edición de SCADA SAINUX.

El objetivo general: Desarrollar un mecanismo de serialización y deserialización que disminuya el tiempo de ejecución, para cargar y guardar la información gestionada en el ambiente de configuración de la Interfaz Hombre Maquina del sistema SCADA SAINUX.

A continuación, se plantean un grupo de tareas que permitirán satisfacer durante el desarrollo de la investigación el objetivo planteado anteriormente, y se muestran de la forma siguiente:

- ✓ Elaborar el marco teórico de la investigación a través del estudio del estado del arte que existe actualmente sobre el tema.
- ✓ Identificar los principales mecanismos de serialización de la información en sistemas SCADA.
- ✓ Caracterizar los mecanismos de serialización de la información en sistemas SCADA.
- ✓ Realizar el levantamiento de requisitos funcionales y no funcionales.
- ✓ Implementar el mecanismo que brinde solución al problema planteado.
- ✓ Realizar pruebas para validar el cumplimiento de los requerimientos.

Métodos científicos de la investigación

Métodos Teóricos

Histórico – Lógico: mediante este método científico se realizó un estudio del estado del arte, sobre los principales mecanismos de serialización que se utilizan en la actualidad prestando especial atención a los mecanismos de serialización binaria.

Analítico – Sintético: el uso del método científico analítico–sintético permitió realizar un estudio por separado de algunos de los mecanismos de serialización, se definió que particularidades presentaban en común y se estableció una series de parámetros, atendiendo principalmente a las características relacionadas con sus objetivos fundamentales, para establecer una comparación entre ellas y tomar los resultados arrojados por dicha comparación, como datos de gran interés para la actual investigación.

Inducción – Deducción: mediante la aplicación del mismo se desarrolló un estudio con las principales herramientas existentes para la serialización y deserialización, revisando sus características propias, basado en estas características se definieron características o cualidades que debe tener o cumplir el sistema que se propone en el presente trabajo.

Método de la modelación: para crear abstracciones con el objetivo de explicar la realidad, se utilizará para la modelación de los diversos diagramas necesarios en cada uno de los flujos de trabajo según la metodología seleccionada.

Métodos Empíricos

Consulta de la información en todo tipo de fuentes: lo cual permite la elaboración del marco teórico de la investigación, y su orientación metodológica.

Observación: mediante el método científico de la observación, se detectó la situación actual existente en el entorno de configuración del HMI del proyecto SCADA SAINUX, en cuanto a las dificultades existentes a la hora de llevar a cabo el proceso de serialización y deserialización de objetos.

Posibles resultados: Obtener la implementación de un mecanismo para realizar la serialización binaria del árbol de proyecto del entorno de configuración del HMI.

El presente documento se estructura en tres capítulos, además de las secciones de Introducción, Conclusiones, Recomendaciones, Referencias Bibliográficas y Anexos.

Capítulo 1. Fundamentación teórica: Definición de los conceptos asociados al dominio de la investigación, sistematización de las herramientas que permiten la serialización y deserialización de objetos. Descripción y caracterización de la metodología, las herramientas y tecnologías empleadas en el desarrollo del sistema.

Capítulo 2. Análisis y Diseño: Diseño de la solución propuesta guiado por la metodología de desarrollo definida, identificación de funcionalidades, descripción de historias de usuario y consideraciones del diseño del sistema.

Capítulo 3. Implementación y Validación de la solución: Se expone la solución que se le dará al problema, a través de un conjunto de artefactos, como el diagrama de componentes y los resultados de las pruebas que se le aplicarán al mismo.

CAPÍTULO 1. **FUNDAMENTACIÓN TEÓRICA**

En el presente capítulo se abordan los principales conceptos y definiciones relacionados con la serialización y deserialización de objetos. Se presentan los tipos más importantes de serialización, las ventajas de realizar serialización y deserialización, las bibliotecas que realizan serialización de objetos para apoyar el funcionamiento del lenguaje C++, los lenguajes que soportan la serialización de forma directa. Además, se define la metodología de desarrollo de software a utilizar, la herramienta y el lenguaje de modelado, el lenguaje de programación y el IDE [5]para realizar la investigación.

1.1. Marco teórico

En las ciencias de la computación la serialización es un proceso de codificación de objetos en un medio de almacenamiento, como puede ser un archivo o un búfer de memoria, con el fin de transmitirlo a través de una conexión en red como una serie de bytes o almacenarlos para su posterior utilización. Muchos lenguajes de programación utilizan la serialización como una forma estándar para almacenar/transmitir objetos en archivos. Este mecanismo permite almacenar en un archivo o enviar a través de una conexión uno o varios objetos con todas sus referencias. La serialización es empleada fundamentalmente como un mecanismo para el almacenamiento de los datos en la programación orientada a objetos y en la comunicación entre componentes (2).

El paradigma de la programación orientada a objetos (POO) está fundamentado en la utilización de objetos y sus interacciones para diseñar aplicaciones y programas informáticos. Utiliza la herencia, abstracción, polimorfismo y encapsulamiento (3) (4).

1.1.1. Descripción del proyecto SCADA SAINUX

El desarrollo científico-técnico alcanzado por la humanidad en las últimas dos centurias ha provocado un aumento del número de productos a fabricar para satisfacer las necesidades de la vida diaria y también un incremento de los estándares de calidad que deben cumplir para que se considere aptos para el consumo. En auxilio de la industria, las tecnologías de la información y las comunicaciones con el desarrollo de potentes sistemas de software han ayudado no solo a incrementar la productividad sino también la calidad de los productos fabricados.

[5] IDE, acrónimo del inglés *Integrated Development Environment*.

Entre los sistemas de software que es imprescindible tener en las industrias actuales se encuentran los sistemas SCADA. Este sistema ayuda a supervisar y mantener un estricto control sobre todos los procesos que ocurren en la industria, de tal manera que se puedan evitar accidentes, perdidas de materias primas por distorsión de los parámetros de industriales e interrupciones en el proceso de fabricación. Además, los sistemas SCADA proveen de valiosa información a sistemas empresariales para la toma de decisiones.

Contribuir con la automatización e informatización de las industrias cubanas es una de las misiones del Centro de Informática Industrial de la Universidad de las Ciencias Informáticas para lo cual desarrolla el sistema SCADA SAINUX. El mismo es un sistema distribuido que cuenta con los siguientes módulos: Comunicaciones, Configuración, Adquisición de Datos, Base de Datos Histórico, Seguridad, Comunicación con Terceros, Modulo de Comunicación OPC y Módulo de Interfaz Hombre-Máquina.

El módulo de HMI del SCADA SAINUX es una aplicación tipo *Desktop* construida utilizando el framework Qt y el lenguaje de programación C++. Esta aplicación es la encargada de la interacción del usuario con el sistema SCADA. Para esto cuenta con diferentes vistas que presentan la información al usuario en formas de sumarios, despliegues, gráficos de tendencia y reportes. Además, a través de las vistas de detalles de puntos, sub canales y dispositivos, se le permite al usuario controlar el proceso mediante el envío de comandos de escritura de puntos, así como de poner o sacar un dispositivo en la recolección de datos (5).

1.1.2. Serialización

La serialización se puede definir como el proceso de convertir el estado de un objeto a un medio que se pueda almacenar o transportar por la red. Durante este proceso, los campos públicos y privados del objeto y el nombre de la clase, incluido el ensamblado que contiene la clase, se convierten a una secuencia de bytes que se escriben en una fuente de datos. Las dos razones más importantes para utilizar la serialización, es conservar el estado de un objeto a los medios de almacenamiento y enviar el valor del objeto de un dominio de aplicación a otro. El estado de un objeto está dado básicamente por el estado de sus campos y serializar un objeto radica en almacenar el estado de sus campos o convertirlos a una secuencia de bytes. Si un objeto determinado a serializar tiene campos que además son objetos, se procede a serializar primero esos objetos. Este es un proceso recursivo que realiza la

serialización de todo un árbol o grafo de objetos, almacenando la información referente a dicho árbol (6) (7).

1.1.3. Deserialización

El complemento de la serialización es la deserialización y es que, al convertir un objeto a una secuencia de bytes, el proceso de deserialización permite la conversión del mismo a su estado original. Este proceso crea una copia exacta del objeto serializado (7) (6).

1.1.4. Principales ventajas que aportan la serialización y la deserialización:

- ✓ Escalabilidad en la aplicación sin importar cuál sea la arquitectura física o lógica.
- ✓ Almacenar o enviar el estado de los objetos a cualquier plataforma o lenguaje.
- ✓ Utilización de protocolos simples para su envío o almacenamiento.
- ✓ Interoperabilidad entre aplicaciones capas o servidores.

1.1.5. Lenguajes de programación que soportan serialización de forma nativa:

Varios lenguajes de programación orientados a objetos soportan la serialización de forma directa, como es el caso de *Python*, *Perl* y *Java*. El lenguaje *Python* provee la serialización mediante módulos que facilitan esta tarea, como marshal, pickle, cPickle y shelve (8).

El lenguaje Perl permite serialización de código incluido dentro de un objeto, el código de las funciones de evaluación se almacena dentro de los objetos como un puntero a código y para serializarlo solo se llama a una función del módulo de serialización (9).

En Java se realiza la serialización mediante la implementación de la interfaz *java.io.serializable*, en caso contrario no se podrá hacer la serialización del objeto. La interfaz *serializable* no tiene métodos, solo es usada para informar a la JVM[6] que un objeto va a ser serializado (10).

1.1.6. Metodologías de serialización

Para implementar la serialización de objetos se utilizan dos metodologías, la intrusiva y la no intrusiva.

- ✓ Intrusiva, se encarga de la modificación de la clase para añadir un método de nombre fijo que serialice la clase, acepta un objeto, así como el número de versión.

[6] JVM, acrónimo del inglés Java Virtual Machine.

- ✓ No intrusiva, se encarga de declarar una función de plantilla global que acepta un número, una instancia de la clase y el número de versión.

En la presente investigación para implementar el mecanismo de serialización se utiliza la metodología no intrusiva. Esta al ser una implementación centralizada, da la posibilidad que, al incluir nuevos componentes o módulos, no haya que cambiar la implementación de la serialización, esto se conoce formalmente como bajo acoplamiento (11).

1.1.7. Tipos de serialización

Existen principalmente dos tipos de serialización de objetos, la serialización de texto plano y la serialización binaria. Para desarrollar un producto se escoge la más recomendada según la necesidad del sistema que se requiere.

Serialización de texto plano.

La codificación en texto plano (ASCII[7]) utiliza 128 valores únicos (0-127), para representar los caracteres alfabéticos, numéricos y de puntuación utilizado habitualmente en inglés, además de una selección de códigos de control que no representan caracteres imprimibles. Por ejemplo, la letra mayúscula 'A' es tiene el código 65 en ASCII, el número '2' es el ASCII 50, el carácter '}' es 125, y el meta carácter retorno de carro es el 13. Los sistemas basados en ASCII utilizan siete bits para representar estos valores de forma digital. Existen 95 códigos imprimibles, de 32 a 126 son conocidos como los caracteres ASCII imprimibles. Como por ejemplo los rangos de letras (A-Z, a-z), números (0-9) y los símbolos '+', '/', '=', entre otros.

Implementaciones de serialización de texto plano:

XML: es el lenguaje de marcado de documentos basado en texto, que contiene los datos estructurados y extensibles. Este lenguaje puede leerse como un texto normal y debido a que es extensible se puede escribir cualquier tipo de información. Los documentos XML pueden contener texto, anotaciones, etc. La serialización XML convierte el tipo de datos a secuencia XML que después se puede convertir a un documento XML para transmitirlo por la red o almacenarlo, en este proceso se toman los tipos de datos complejos (u objetos) y se transforman a una secuencia XML. Esto resulta útil cuando se desea proporcionar o consumir datos sin restringir la aplicación que utiliza los datos. Como XML es un

[7] API, acrónimo del inglés *Application Programming Interface*.

estándar abierto, es una opción atractiva para compartir datos a través de la web. Este tipo de serialización es ideal para cartografía rápida (12).

Qt Xml: anteriormente era el módulo de Qt para la serialización de datos de XML, a partir de la versión 5 de Qt, este módulo queda sin soporte, lo que dio lugar que las clases involucradas se movieran al módulo Qt Core. La API es muy sencilla, la cual constituye escribir o leer en un QIODevice (13).

JSON: es un formato de texto ligero para el intercambio de datos. JSON[8] es un subconjunto de la notación literal de objetos de JavaScript, aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato independiente de algún lenguaje en específico. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval, lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX[9], debido a la ubicuidad de *JavaScript* en casi cualquier navegador web (14).

YAML: es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl (15).

Serialización binaria.

La serialización binaria toma el tipo de datos y lo convierte a una secuencia binaria, en este proceso los tipos de datos complejos (u objeto) son transformados a una secuencia binaria, que es un estado resistente y facilita su transportación. La serialización binaria conserva la información del tipo de objeto en el flujo de datos generados, esto significa que cuando el objeto se deserializa, el objeto creado es una copia exacta del original. La serialización binaria conserva la fidelidad de tipos, lo que resulta útil para conservar el estado de un objeto entre distintas llamadas a una aplicación. La serialización binaria es muy recomendada para la comunicación entre componentes (16).

Implementaciones de serialización binaria:

Serialización con Boost: es un conjunto de bibliotecas de software libre desarrolladas para incrementar las funcionalidades del lenguaje de programación C++. Estas bibliotecas son útiles en

[8] JSON, acrónimo del inglés *JavaScrip Object Notation* (aunque ha sido adoptado como estándar por muchos lenguajes).
[9] AJAX, acrónimo del inglés *Asynchronous JavaScript And XML*.

muchas aplicaciones. Su licencia permite utilizarlas en cualquier tipo de proyecto, ya sean comerciales o no comerciales. Brinda serialización automática de estructuras de la STL[10] (es un conjunto determinado de estructuras de datos y algoritmos que integran la librería estándar C++, es decir conjunto de recursos que ya están implementados en este lenguaje y su elevado nivel de abstracción). Ofrece versionado, o sea, cuando se realiza el proceso de serialización y se crean archivos sobre la base de este, se debe estar seguro que esos archivos sean compatibles con otras aplicaciones (16).

Protocol Buffers: es un lenguaje independiente a la plataforma, implementa un mecanismo eficiente y automatizado utilizado para la serialización de datos estructurados, se utiliza en protocolos de comunicación y almacenamiento de datos. Es un mecanismo flexible, eficiente, automatizado similar a XML o JSON, pero genera datos más pequeños, es más rápido y más simple. Se define cómo desea que sus datos sean estructurados de una vez, se puede usar el código fuente generado para escribir y leer con facilidad los datos hacia y desde una variedad de flujos de datos y usando una variedad de lenguajes de programación. Puede actualizar las estructuras de datos sin romper los programas implementados que han sido compilados con el formato "viejo". Google utiliza el *Protocol Buffers* para casi la totalidad de sus llamadas a procedimientos remotos (17).

MessagePack: es un formato de serialización binaria eficiente. Se le permite intercambiar datos entre varios lenguajes de programación, al igual que lo hace JSON, pero a diferencia de este es más rápido y más pequeño. Los números enteros pequeños se codifican en un solo *byte*, y las cadenas cortas típicas requieren de sólo un *byte* adicional además de las propias (18).

QDataStream: ofrece la serialización de datos binarios a un QIODevice. Un flujo de datos es una corriente binaria de información codificada que es 100% independiente del sistema operativo, CPU o del *"endian" host*. El flujo de datos coopera estrechamente con un QIODevice (19) (20).

S11N.Net: es un proyecto código abierto centrado en la serialización de objetos genéricos (es decir, de persistencia de objeto) en el lenguaje de programación C ++ (21).

RAW es un mecanismo para escribir los datos a partir de la toma de espacios de memoria con los datos en crudo. Es uno de los mecanismos más eficientes, pero presenta graves problemas de portabilidad (22).

[10] STL, acrónimo del inglés *Standard Template Library*.

Para la selección de la biblioteca de serialización se realiza un *test benchmark* (23) los parámetros a considerar en el resultado son el tiempo de serialización y el espacio ocupado por el *buffer* serializado, con el cual se llega a los siguientes resultados.

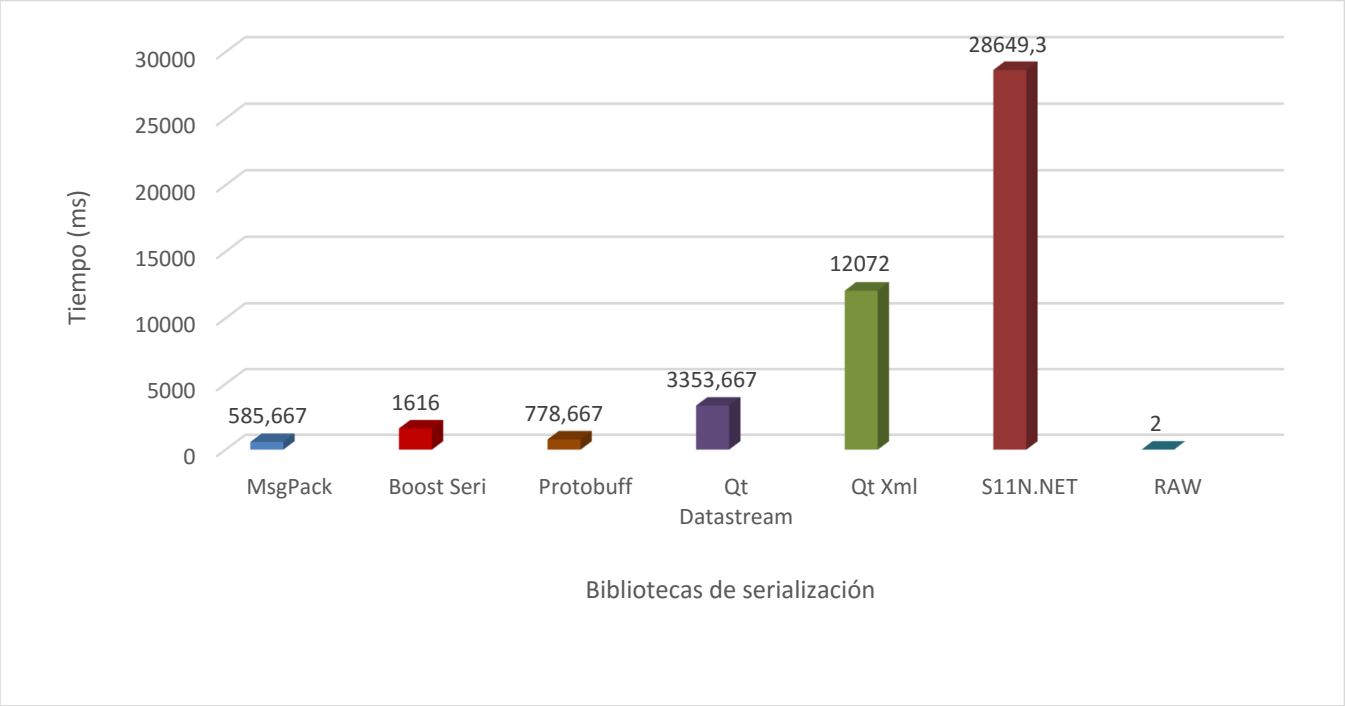


Figura 1. Pruebas a las bibliotecas por tiempo.

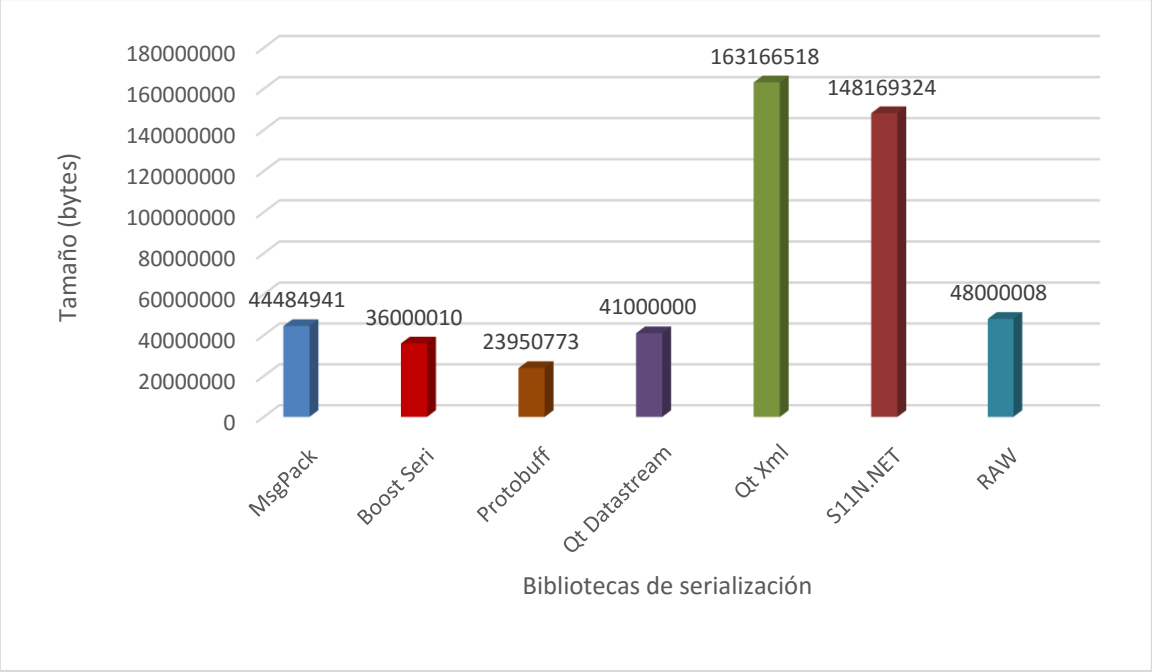


Figura 2. Pruebas a las bibliotecas por tamaño.

Basándose en estos resultados y el problema planteado en la introducción se selecciona la biblioteca *QdataStream* del módulo *Qt Core* del *framework Qt,* ya que el principal objetivo es disminuir el tiempo de serialización de un proyecto del ambiente de configuración de *HMI.*

1.2. Metodologías de Desarrollo

En el proceso de construcción de un software las metodologías de desarrollo de software constituyen un pilar fundamental evitando que este vaya rumbo al fracaso. Se definen como un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar su trabajo. Una metodología puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto, pero no cómo hacerlo. La metodología indica cómo hay que obtener los distintos productos parciales y finales. Se han desarrollado en los últimos años dos corrientes en lo referente a los procesos de desarrollo, los llamados métodos (o metodologías) pesados y los ligeros (o ágiles).

1.2.1. Metodologías pesadas o tradicionales

Las metodologías pesadas o tradicionales son más eficaces y necesarias cuanto mayor es el proyecto que se pretenda realizar respecto a tiempo y recursos que son necesarios emplear, se centran en la

definición detallada de los procesos y tareas a realizar, herramientas a utilizar, y requiere una extensa documentación, ya que pretenden prever todo de antemano.

Dentro del grupo de metodologías que encierra esta clasificación la más utilizada a nivel mundial es RUP[11]. Las principales características de esta metodología para el desarrollo de software es realizar un proceso iterativo e incremental, centrado en la arquitectura y guiado por casos de uso. Permite llevar un control de cambios estricto y divide el proceso de desarrollo en cuatro fases por las cuales se transita varias veces: inicio, elaboración, desarrollo y transición (24).

1.2.2. Metodologías ágiles o ligeras

Las metodologías ágiles se encargan de valorar al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados, priorizan la creación del software ante la generación excesiva de documentación, el cliente está en todo momento colaborando en el proyecto y la capacidad de respuesta ante un cambio realizado es más importante que el seguimiento estricto de un plan.

Dentro de las metodologías ágiles se encuentra el Proceso Unificado Ágil o AUP[12] la cuál es una versión simplificada de RUP. AUP dispone de cuatro fases de trabajo al igual que RUP: inicio, elaboración, construcción y transición. Abarca siete flujos de trabajo, cuatro ingenieriles y tres de apoyo: Modelado, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyectos y Ambiente respectivamente. El Modelado agrupa los tres primeros flujos de RUP (Modelado del negocio, Requerimientos, Análisis y Diseño) (24).

1.2.3. Variación AUP – UCI

En la UCI se realizó una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. Las fases que propone son:

- ✓ Inicio: Se llevan a cabo las actividades relacionadas con la planificación del proyecto.
- ✓ Ejecución: Se ejecutan las actividades requeridas para desarrollar el software (incluye entre otras actividades el ajuste de los planes del proyecto, el modelado del negocio, obtención de requisitos, implementación, liberación del producto.

[11] RUP, acrónimo del inglés *Rational Unified Process*.
[12] AUP, acrónimo del inglés *Agile Unified Process*.

- ✓ Cierre: Se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales del cierre de proyecto

Las disciplinas que propone AUP-UCI son: Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas Internas, de Liberación y Aceptación.

La selección de la metodología a utilizar se hace en base a las características del equipo y las necesidades específicas de la situación. Para elegir una metodología de desarrollo de software se debe tener en cuenta dos factores fundamentales: el tipo de proyecto que se desea desarrollar y el tiempo que se dispone para desarrollar el mismo. A partir del análisis realizado a las metodologías pesadas y ligeras y las necesidades del proyecto, se propone utilizar la metodología AUP en la variación de AUP-UCI en el escenario 4, por ser la que más se adapta al proyecto a desarrollar y a las condiciones de trabajo.

1.3. Herramientas y lenguajes

1.3.1. Unified Modeling Language 2.0 (UML)

UML es un lenguaje, que permite modelar, analizar y diseñar sistemas orientados a objetos. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Ofrece un estándar para describir un panorama del sistema modelo, incluyendo aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables y aspectos conceptuales como los procesos de negocios y funciones del sistema. “UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software” (25).

UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Debido a que UML es un lenguaje, cuenta con reglas para combinar tales elementos y permite la modelación de sistemas con tecnología orientada a objetos (25).

1.3.2. Herramienta *Case: Visual Paradigm*

Visual Paradigm (26) es una herramienta que ofrece un entorno de creación de diagramas usando las notaciones UML 2.0, con un diseño centrado en casos de uso y enfocado además al negocio, lo cual genera un software de alta calidad. Esta herramienta fue creada para el ciclo completo de desarrollo de software, permitiendo la captura de requisitos, análisis, diseño e implementación. También proporciona

características tales como generación del código, ingeniería inversa y generación de informes. Permite escribir toda la especificación de un caso de uso sin necesidad de utilizar una herramienta externa como editor de texto, utilizando plantillas que se encuentran definidas, o que pueden ser creadas por los usuarios. Está diseñado para usuarios interesados en sistemas de software de gran escala, apoya los estándares más recientes de la notación UML. A pesar de todo es importante destacar que la licencia de Visual Paradigm es muy restringida. Debido a las funcionalidades que brinda Visual Paradigm, al permitir crear diagramas usando las notaciones UML y ser una herramienta multiplataforma que se integra fácilmente con varios IDE, se decidió usar como herramienta durante el proceso de modelado.

1.3.3. Lenguaje de programación C++

El lenguaje de programación C++ (6) es muy utilizado para el desarrollo de sistemas, permite trabajar tanto a alto como a bajo nivel. Añade utilidades específicas para la programación orientada a objetos y por su potencia y su eficiencia se ha clasificado como uno de los lenguajes más empelados. Por ser multiparadigma soporta la programación orientada a objetos, programación genérica, programación estructurada, entre otros. Una de las particularidades de C++ es la posibilidad de redefinir los operadores o sea la sobrecarga de operadores, y poder crear nuevos tipos que se comporten como tipos fundamentales (27) (28) (29).

1.3.4. Entorno de desarrollo Integrado Qt Creator

Qt Creator (30) ha sido desarrollado para ser un IDE multiplataforma adaptado a las necesidades de los desarrolladores de Qt. Qt Creator se ejecuta en los sistemas operativos de escritorio Windows, Linux/X11 y Mac OS X y permite a los desarrolladores crear aplicaciones para múltiples escritorios y plataformas de dispositivos móviles.

Estas son algunas de las características clave de "Qt Creator":

- ✓ Editor de código sofisticado: El editor de código avanzado de Qt Creator ofrece compatibilidad con la edición del lenguaje C++, QML (JavaScript), entre otros, además ayuda sensible al contexto.
- ✓ Diseñadores integrados de interfaz de usuario: Qt Creator ofrece dos editores visuales integrados, Qt Designer para la creación de interfaces de usuario de widgets Qt y *Qt Quick Designer* para el desarrollo de interfaces de usuario animadas con el lenguaje QML.

- ✓ Administración de proyectos y versiones: Independientemente de si importas un proyecto existente o creas uno desde cero, Qt Creator genera todos los archivos necesarios. Es compatible con *qmake, qbs y CMake*.
- ✓ *Target* de ordenadores de escritorio y portátiles: Qt Creator ofrece compatibilidad con la creación y ejecución de aplicaciones Qt para ordenadores de escritorio y dispositivos móviles. La configuración generada permite cambiar fácilmente entre los destinos de generación.

1.4. Conclusiones parciales

Con la realización del presente capítulo han quedado definidos los conceptos más importantes para el correcto entendimiento de los términos usados en la presente investigación. El análisis a las soluciones similares sirvió para reconocer la existencia de algunas bibliotecas de serialización tanto binaria como de texto plano, así como su desempeño a la hora de serializar y/o deserializar objetos, permitiendo determinar las ventajas de su utilización en el proceso de desarrollo. AUP – UCI como metodología de desarrollo, *QdataStream* como biblioteca de serialización, UML, C++ como lenguaje de modelado y programación respectivamente y Visual Paradigm como herramienta CASE[13].

[13] CASE, acrónimo del inglés *Computer Aided Software Engineering*.

CAPÍTULO 2. ANÁLISIS Y DISEÑO

En el presente capítulo se realiza una propuesta de solución que va a estar conformada por la identificación y descripción de los requisitos funcionales y no funcionales que debe cumplir el sistema a implementar, además de los artefactos generados durante las fases de Análisis y Diseño de la herramienta.

2.1. Propuesta de solución

El proyecto SCADA SAINUX realiza la serialización y deserialización de los recursos que se gestionan en él. En la implementación inicial se salvaban los proyectos en formato XML y no existían tantos objetos a serializar como la cantidad actual, por lo que esto no representaba un problema en aquel entonces. Con el desarrollo del proyecto, durante el tiempo, fue aumentando la cantidad de recursos a serializar, esto conllevó a que la implementación inicial comenzara a presentar problemas de rendimiento, como ejemplo mencionar que se llegó a tener algunos proyectos que demoraban entre 15 y 20 minutos abriendo. Por ello se propone como solución crear una arquitectura extensible mediante el patrón de diseño *Abstract Factory*, de manera que permita incorporar varios mecanismos de serialización, dentro de los mecanismos incorporados se debe proveer la anterior implementación XML y otra de implementación alternativa que resuelva los problemas de rendimiento anteriormente mencionados. Esta implementación alternativa se realiza a través del mecanismo de serialización *QDataStream* del módulo *Core* del framework *Qt*.

2.2. Requisitos del sistema

Para lograr que el desarrollo de un software tenga éxito, es esencial comprender perfectamente los requisitos del software. La captura de los requisitos va a influenciar en todo el proceso de desarrollo del software, repercutiendo en el resto de las fases de desarrollo. Una definición eficiente de los requisitos funcionales y no funcionales, permitirá al equipo de desarrollo reducir los riesgos que se puedan presentar durante la implementación del sistema y se puede determinar lo que el sistema debe hacer. Además, de establecer aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes del mismo (31).

2.2.1. Técnicas para la captura de requisitos

Para realizar una eficiente captura de requisitos respondiendo a las necesidades del cliente se utilizan las siguientes técnicas de captura de requisitos:

Tormenta de ideas

Mediante esta técnica de trabajo en grupo se pretende obtener el mayor número de ideas o soluciones a cuestiones planteadas en el menor tiempo posible, aprovechando la capacidad creativa de las personas. Se emplea fundamentalmente cuando se requieren soluciones creativas e ideas innovadoras, después de agotar los esfuerzos individuales sin obtener resultados satisfactorios.

Observación de sistemas semejantes

Otra técnica de extracción de requisitos es realizar el estudio de soluciones semejantes. Probablemente no permitan obtener los requisitos directamente, pero sí puede dar una buena base para saber qué se pretende buscar y obtener una colección de requisitos de partida que permita agilizar la toma de requisitos.

Mediante el estudio y análisis de otros programas que serializan datos, como por ejemplo QGIS (32), la versión existente de HMI Editor y los ejemplos que vienen con el *framework* Qt, se pudo hallar características o funcionalidades comunes de estos, las cuáles pueden ser utilizadas y, por tanto, brindan una guía para la captura de los requisitos funcionales correspondientes para la serialización del árbol de proyecto de HMI Editor.

QGIS (anteriormente conocido como Quantum GIS[14]) es un sistema *desktop* multiplataforma libre y de código abierto (aunque actualmente se está trabajando en una versión para Android) de información geográfica (GIS) de aplicaciones que proporciona la visualización de datos, edición y análisis.

Funcionalidad

Al igual que en otros sistemas de información geográfica de software, QGIS permite a los usuarios crear mapas con muchas capas con diferentes proyecciones de mapas. Los mapas pueden ser montados en diferentes formatos y para distintos usos. QGIS permite que los mapas que se componen de capas de mapa de bits o vectoriales. Típico de este tipo de software, los datos vectoriales se almacenan, ya sea como punto, línea o polígono-espacial. Diferentes tipos de imágenes de los mapas de bits son compatibles, y el software puede georeferenciar imágenes.

Funciones avanzadas

[14]GIS, acrónimo del inglés *Geographical Information System*.

QGIS se integra con otros paquetes de SIG de código abierto, incluyendo *PostGIS*, *Grass*, y *MapServer* para dar a los usuarios una amplia funcionalidad. Las extensiones escritas en Python o C++ amplian las capacidades de QGIS. Las extensiones pueden codificar geográficamente mediante la API de codificación geográfica de Google, realizar geoprocesamiento utilizando *fTools*, que son similares a las herramientas estándar que se encuentran en *ArcGIS*, y la interfaz con bases de datos *PostgreSQL* / *PostGIS*, *SpatiaLite* y *MySQL*.

La alta complejidad de esta herramienta permite tomarla como un caso de estudio ideal ya que al igual que HMI cuenta con una gran cantidad de recursos a serializar para permitir las funcionalidades de guardar y abrir un proyecto de forma local.

2.2.2. Requisitos Funcionales

Luego de analizar la problemática planteada y utilizando las técnicas para la captura de requisitos descritos se han obtenido los siguientes requisitos funcionales:

Tabla 1. Requisitos funcionales.

Nº	Nombre	Descripción	Complejidad
RF1	Serializar los tipos de datos del *framework* Qt.	Los tipos de datos que define el *framework* Qt.	Media
RF1.1	Serializar la clase "QPen".	La clase define como un *QPainter* debe dibujar líneas y contornos de formas. Un *QPen* tiene *style(), width(), brush(), capStyle()* y *joinStyle()*.	Media
RF1.2	Serializar la clase "QBrush".	La clase define el patrón de relleno de formas dibujadas por *QPainter*. Un *Brush* tiene un estilo, un color, un gradiente y una textura.	Media

RF1.3	Serializar la clase "QFont".	La clase especifica una fuente usada para dibujar texto.	Media
RF1.4	Serializar la clase "QSize".	La clase define el tamaño de un objeto en dos dimensiones utilizando precisión de punto entero. Un *QSize* se especifica por una anchura y una altura.	Media
RF1.5	Serializar la clase "QSizeF".	La clase define el tamaño de un objeto bidimensional usando la precisión de punto flotante.	Media
RF1.6	Serializar la clase "QPoint".	La clase define un punto en el plano utilizando precisión entera. Un *QPoint* se especifica mediante una coordenada *x* y una coordenada *y*.	Media
RF1.7	Serializar la clase "QPointF".	La clase define un punto en el plano usando la precisión de punto flotante.	Media
RF1.8	Serializar la clase "QColor".	La clase proporciona colores basados en valores *CMYK*, *RGB* o *HSV*. Un *QColor* normalmente se especifica en términos de componentes *RGB* (rojo, verde y azul), pero también es posible especificar en términos de	Media

		componentes *HSV* (matiz, saturación y valor) y *CMYK* (cian, magenta, amarillo y negro).	
RF1.9	Serializar la clase "QRect".	La clase define un rectángulo en el plano utilizando precisión entera. Un *QRect* se expresa normalmente como una esquina superior izquierda y un tamaño.	Media
RF1.10	Serializar la clase "QRectF".	La clase define un rectángulo en el plano usando la precisión de punto flotante.	Media
RF1.11	Serializar la clase "QTime".	La clase ofrece funciones de tiempo de reloj. Un objeto *QTime* contiene una hora del reloj, es decir, el número de horas, minutos, segundos y milisegundos desde la medianoche.	Media
RF1.12	Serializar la clase "QList".	La clase es una clase de plantilla que proporciona listas. *QList<T>* es una de las clases de contenedores genéricos de *Qt*. Se almacena una lista de valores y proporciona acceso basado en el índice de rápido, así como las inserciones y eliminaciones rápidas.	Media

RF1.13	Serializar la clase "QString".	La clase proporciona una cadena de caracteres *Unicode*. *QString* almacena una cadena de *QChar* 16 bits, donde cada *QChar* corresponde un carácter Unicode 4.0. Los caracteres *Unicode* con valores de código por encima de 65535 se almacenan utilizando pares suplentes, es decir, dos *QChar* consecutivos.	Media
RF1.14	Serializar la clase "QStringList".	La clase proporciona una lista de cadenas. *QStringList* hereda de *QList<QString>.* Al igual que *QList, QStringList* se comparte de forma implícita. Proporciona acceso rápido basado en el índice, así como las inserciones y eliminaciones rápidas.	Media
RF2	Serializar las entidades del negocio.	Las entidades del negocio que define el usuario.	Alta
RF2.1	Serializar la clase "ColorForState".	La clase que define el estado de una válvula en una tubería de petróleo. Está representado por un color y una descripción.	Alta
RF2.2	Serializar la clase	La clase es una estructura de datos lineales que contiene	Alta

	"ObjectsCollection".	varios punteros. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	
RF2.3	Serializar la clase "ColorCollection".	La clase es una estructura de datos lineales que contiene varios colores de la clase *QColor*. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	Alta
RF2.4	Serializar la clase "VariantCollection"	La clase es una estructura de datos lineales que contiene varios *QVariant*. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	Alta
RF3	Serializar los tipos de datos nativos del lenguaje.	Los tipos de datos nativos del lenguaje que están definidos por el estándar de la ISO del lenguaje C++.	Baja
RF3.1	Serializar el tipo de dato "bool".	Es el tipo de dato *booleano* del lenguaje de programación C++.	Baja
RF3.2	Serializar el tipo de dato "unsigned int".	Es el tipo de dato entero sin signo del lenguaje de programación C++.	Baja
RF3.3	Serializar el tipo de dato "unsigned long long".	Es el tipo de dato *long long* sin signo del lenguaje de programación C++.	Baja

RF3.4	Serializar el tipo de dato "double".	Es el tipo de dato *double* del lenguaje de programación C++.	Baja
RF3.5	Serializar el tipo de dato "char".	Es el tipo de dato *char* del lenguaje de programación C++.	Baja
RF4	Deserializar los tipos de datos del *framework* Qt.	Los tipos de datos que define el *framework* Qt.	Media
RF4.1	Deserializar la clase "QPen"	La clase define como un *QPainter* debe dibujar líneas y contornos de formas. Una *QPen* tiene *style(), width(), brush(), capStyle()* y *joinStyle().*	Media
RF4.2	Deserializar la clase "QBrush".	La clase define el patrón de relleno de formas dibujadas por *QPainter*. Un *Brush* tiene un estilo, un color, un gradiente y una textura.	Media
RF4.3	Deserializar la clase "QFont".	La clase especifica una fuente usada para dibujar texto.	Media
RF4.4	Deserializar la clase "QSize".	La clase define el tamaño de un objeto en dos dimensiones utilizando precisión entera. Un *QSize* se especifica por una anchura y una altura.	Media
RF4.5	Deserializar la clase "QSizeF".	La clase define el tamaño de un objeto bidimensional usando la precisión de punto	Media

		flotante.	
RF4.6	Deserializar la clase "QPoint".	La clase define un punto en el plano utilizando precisión entera. Un *QPoint* se especifica mediante una coordenada *x* y una coordenada *y*.	Media
RF4.7	Deserializar la clase "QPointF".	La clase define un punto en el plano usando la precisión de punto flotante.	Media
RF4.8	Deserializar la clase "QColor"	La clase proporciona colores basados en valores *CMYK*, *RGB* o *HSV*. Un *QColor* normalmente se especifica en términos de componentes *RGB* (rojo, verde y azul), pero también es posible especificar en términos de componentes *HSV* (matiz, saturación y valor) y *CMYK* (cian, magenta, amarillo y negro).	Media
RF4.9	Deserializar la clase "QRect".	La clase define un rectángulo en el plano utilizando precisión entera. Un *QRect* se expresa normalmente como una esquina superior izquierda y un tamaño.	Media
RF4.10	Deserializar la clase "QRectF"	La clase define un rectángulo en el plano usando la	Media

		precisión de punto flotante.	
RF4.11	Deserializar la clase "QTime"	La clase ofrece funciones de tiempo de reloj. Un objeto *QTime* contiene una hora del reloj, es decir, el número de horas, minutos, segundos y milisegundos desde la medianoche.	Media
RF4.12	Deserializar la clase "QList".	La clase es una clase de plantilla que proporciona listas. *QList<T>* es una de las clases de contenedores genéricos de *Qt*. Se almacena una lista de valores y proporciona acceso basado en el índice de rápido, así como las inserciones y eliminaciones rápidas.	Media
RF4.13	Deserializar la clase "QString".	La clase proporciona una cadena de caracteres *Unicode*. *QString* almacena una cadena de *QChar* 16 bits, donde cada *QChar* corresponde un carácter Unicode 4.0. Los caracteres *Unicode* con valores de código por encima de 65535 se almacenan utilizando pares suplentes, es decir, dos *QChar* consecutivos.	Media

RF4.14	Deserializar la clase "QStringList".	La clase proporciona una lista de cadenas. *QStringList* hereda de *QList<QString>.* Al igual que *QList, QStringList* se comparte de forma implícita. Proporciona acceso basado en el índice de rápido, así como las inserciones y eliminaciones rápidas.	Media
RF5	Deserializar las entidades del negocio.	Las entidades del negocio que define el usuario.	Alta
RF5.1	Deserializar la clase "ColorForState".	La clase que define el estado de una válvula en una tubería de petróleo. Está representado por un color y una descripción.	Alta
RF5.2	Deserializar la clase "ObjectsCollection".	La clase es una estructura de datos lineales que contiene varios punteros. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	Alta
RF5.3	Deserializar la clase "ColorCollection".	La clase es una estructura de datos lineales que contiene varios colores de la clase *QColor*. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	Alta

RF5.4	Deserializar la clase "VariantCollection"	La clase es una estructura de datos lineales que contiene varios *QVariant*. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	Alta
RF6	Deserializar los tipos de datos nativos del lenguaje.	Los tipos de datos nativos del lenguaje que están definidos por el estándar de la ISO del lenguaje C++.	Baja
RF6.1	Deserializar el tipo de dato "bool".	Es el tipo de dato booleano del lenguaje de programación C++.	Baja
RF6.2	Deserializar el tipo de dato "unsigned int".	Es el tipo de dato entero sin signo del lenguaje de programación C++.	Baja
RF6.3	Deserializar el tipo de dato "unsigned long long".	Es el tipo de dato *long long* sin signo del lenguaje de programación C++.	Baja
RF6.4	Deserializar el tipo de dato "double".	Es el tipo de dato *double* del lenguaje de programación C++.	Baja
RF6.5	Deserializar el tipo de dato "char".	Es el tipo de dato *char* del lenguaje de programación C++.	Baja

2.2.3. Requisitos no funcionales

Los requisitos no funcionales hacen relación a las características del sistema que aplican de manera general como un todo, más que a rasgos particulares del mismo. Un requisito no funcional especifica los criterios

que se deben usar para juzgar el funcionamiento de un sistema y no su comportamiento específico. A continuación, se describe el requisito no funcional que mayor relación guarda con el problema de investigación, los restantes requisitos no funcionales se encuentran descritos en los anexos.

Tabla 2. RNF- Eficiencia/Tiempo.

Atributo de Calidad	Eficiencia.
Sub-atributos/Sub-características	Tiempo.
Objetivo	Garantizar que el proceso de serialización y deserialización sea más eficiente en cuanto al tiempo consumido.
Origen	Humano.
Artefacto	Aplicación.
Entorno	La aplicación está funcionando correctamente.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
1. a Serializar y deserializar recursos.	
La aplicación debe serializar y deserializar en un tiempo menor al proceso anterior.	La aplicación serialización y deserialización se realiza en un tiempo menor.
Medida de respuesta	
Disponibilidad de la aplicación.	

2.3. Historias de Usuario

Las Historias de Usuario (HU) son la técnica utilizada en AUP-UCI en su escenario 4 para especificar los requisitos del software y constituyen el artefacto más importante de esta metodología. Se tratan de tarjetas en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las HU es muy dinámico y flexible, en cualquier momento las HU pueden desecharse, remplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada HU debe ser comprensible y delimitada para que los programadores puedan

implementarlas en unas semanas (33). A continuación, se muestra la descripción de una historia de usuario de prioridad alta, las demás de esta prioridad se pueden apreciar en los anexos.

Tabla 3. HU1-Serializar "ColorForState".

Historia de usuario	
Número: 2.1	**Nombre del requisito:** Serializar la clase "ColorForState"
Programador: Alvaro	**Iteración:** 1
Prioridad: Alta	**Tiempo estimado:** 24 horas
Riesgo de desarrollo: Baja	**Tiempo real:** 24 horas
Descripción: Permite serializar un objeto de la clase "ColorForState". La clase que define el estado de una válvula en una tubería de petróleo. Está representado por un color y una descripción.	

2.4. Diagrama de paquetes

Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Los paquetes se pueden construir para representar relaciones tanto físicas como lógicas (34). La propuesta de solución se desarrolla en el paquete *Util* del proyecto HMI (ver Figura 3).

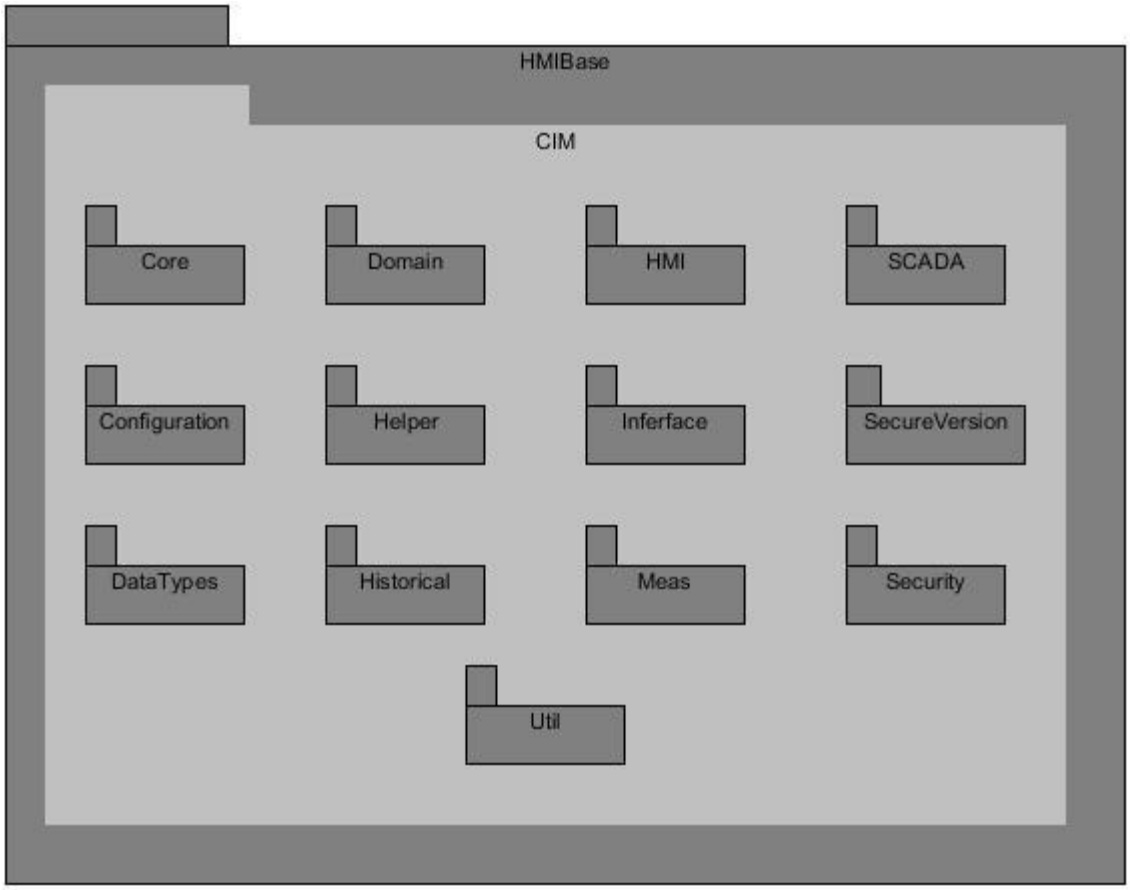


Figura 3. Diagrama de paquetes.

2.5. Diagrama de clases del diseño

El diagrama de clases muestra las definiciones de las clases que se implementarán, las relaciones entre ellas y los métodos que cada una debe definir (25). El siguiente diagrama de clases (ver Figura 4) representa el proceso de serialización/deserialización de objetos y luego se puede apreciar en la Tabla 4 la descripción de las clases.

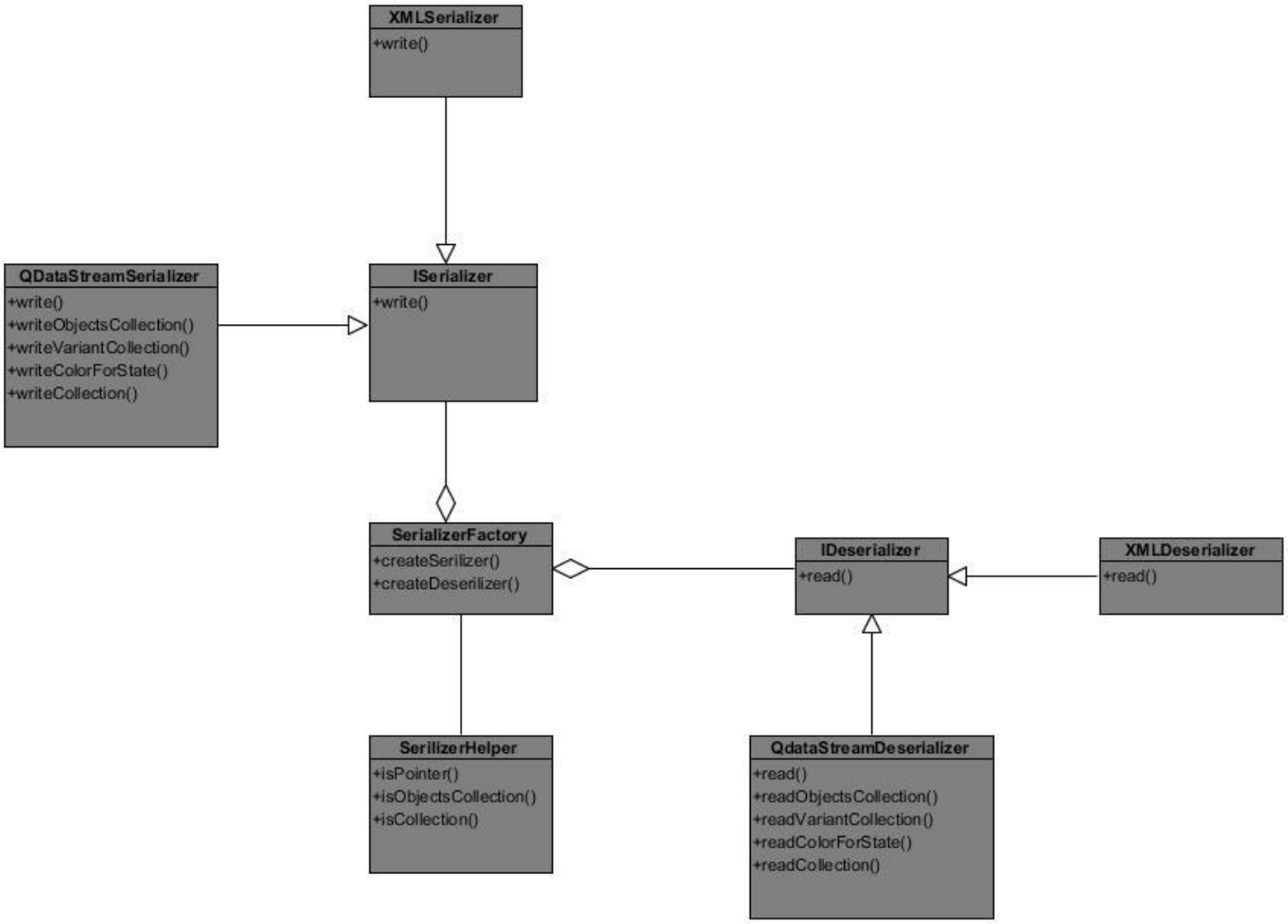


Figura 4. Diagrama de clases del diseño.

Tabla 4. Descripción de las clases.

Clases	Descripción
XMLSerializer	Esta clase es la encargada de serializar en formato XML.
QDataStreamSerializer	Esta clase es la encargada de serializar en formato binario QdataStream.

ISerializer	Esta clase es la interfaz genérica de serialización.
SerializerFactory	Es la clase fábrica encargada de crear el serializador adecuado.
SerilizerHelper	Es un *wraper* que facilita el uso de las clases de serialización/deserialización.
IDesSerializer	Esta clase es la interfaz genérica de deserialización.
QdataStreamDesSerializer	Esta clase es la encargada de deserializar en formato binario QdataStream.
XMLDesSerializer	Esta clase es la encargada de deserializar en formato XML.

2.6. Patrones de diseño

Los patrones de diseño son soluciones a problemas repetidos en la construcción de software, y en ocasiones pueden incluir sugerencias para aplicar estas soluciones en diversos entornos. En la implementación de las clases se utilizó el patrón de diseño *Factory* (35).

Factory: se define un objeto "fábrica" para crear los objetos. Los objetos fábrica tienen varias ventajas: separan la responsabilidad de la creación compleja en objetos de apoyo *(helper)* cohesivos, ocultan la lógica de creación potencialmente compleja y permiten introducir estrategias para mejorar el rendimiento de la gestión de la memoria, como objeto caché o de reciclaje. En la siguiente figura se puede apreciar como los métodos de la factoría devuelven objetos de tipo de interfaz en vez de una clase, de manera que la factoría puede devolver cualquier implementación de interfaz (35).

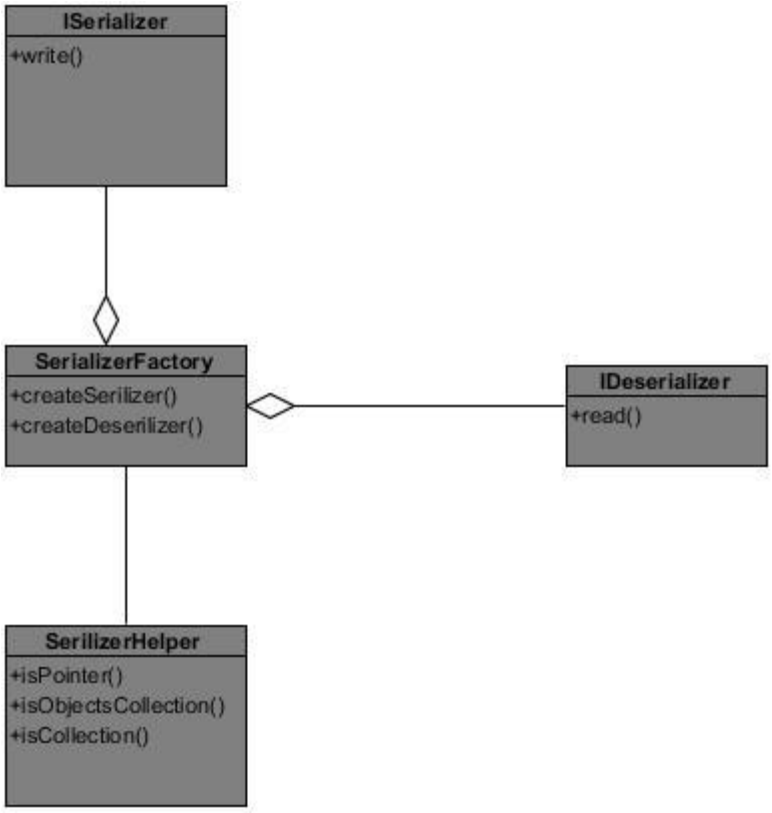


Figura 5. Patrón Factory.

2.7. Conclusiones parciales

En el presente capítulo se mostraron los artefactos generados durante el diseño. Se incluye una breve descripción de los mismos, con el objetivo de alcanzar perfectamente los requisitos del software; posibilitando la correcta transformación de los mismos a un diseño que indique cómo debe ser implementado el software. También se describieron los patrones utilizados para el diseño propuesto y se obtuvo la implementación del mecanismo de serialización y deserialización de objetos.

CAPÍTULO 3. IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

Partiendo del resultado del análisis y diseño, en el presente capítulo se generan los productos de trabajo referentes a la fase de implementación y prueba del software. Se especifican los casos de pruebas aplicados a la solución desarrollada para validar su correcto funcionamiento.

3.1. Implementación

En muchas ocasiones, los sistemas de cómputo son implementados por varios programadores, la adopción inicial de un único estilo de codificación constituye uno de los factores de mayor peso en la calidad, rendimiento, legibilidad y capacidad de mantenimiento del producto final. A continuación, se describe el estándar de codificación a tener en cuenta en la solución a desarrollar.

3.1.1. Estándar de codificación.

Los estándares de codificación son especificaciones o estilos que establecen la forma de generar el código de las aplicaciones informáticas. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico (36) (37).

Declaración de variables

- Declarar cada variable en una línea separada.
- Evitar nombres cortos o sin sentido, los nombres de variables de carácter individual son sólo bien para los contadores y los temporales, donde el propósito de la variable es obvio.
- Espere al declarar una variable hasta que sea necesario.
- Las variables y funciones comienzan con una letra minúscula. Cada palabra consecutiva en el nombre de una variable comienza con una letra mayúscula.
- Evitar abreviaturas.
- Los acrónimos son *camel-cased* (por ejemplo, *QXmlStreamReader*, no *QXMLStreamReader*).

Espacio en blanco

- Utilice líneas en blanco para agrupar bloques lógicos.
- Siempre utilizar sólo una línea en blanco.
- Siempre use un solo espacio después de una palabra clave y antes de una llave.

- Para punteros o referencias, utilice siempre un espacio entre el tipo y '*' o '&', pero no hay espacio entre el "*" o "&" y el nombre de la variable.
- Rodear los operadores binarios con espacios.
- No hay espacio después de un *cast*.
- Evitar los *cast* al estilo C siempre que sea posible.
- No ponga varias instrucciones en una línea.
- De manera general, utilice una nueva línea para el cuerpo de control de flujo.

Llaves

- La llave de apertura va en la misma línea que el inicio de la declaración. Si la llave de cierre es seguida por otra palabra clave, entra en la misma línea.
- Excepción: las implementaciones de función y las declaraciones de clase siempre tienen la llave izquierda en el inicio de una línea.
- Utilizar llaves sólo cuando el cuerpo de una sentencia condicional contiene más de una línea.
- Excepción 1: Use las llaves también si la declaración padre cubre varias líneas.
- Excepción 2: Simetría de las llaves: El uso de las llaves también en *if-then-else* o bien *if-code* o *else-code* abarca varias líneas.
- Utilizar llaves cuando el cuerpo de una sentencia condicional está vacío.

Paréntesis

- Utilice paréntesis para agrupar expresiones.

Sentencias switch

- Las etiquetas de caso están en la misma columna que el *'switch'*.
- Cada caso debe tener una sentencia *break* (o retorno) al final o un comentario para indicar que no hay ningún *break* intencional, a menos que otro caso sigue inmediatamente.

Sentencias saltos ('break', 'continue', 'return', and 'goto')

- No ponga *'else'* después de las declaraciones de saltos.
- Excepción: Si el código es inherentemente simétrico, se permite el uso de *'else'* para visualizar la simetría.

Saltos de línea

- Mantenga las líneas más cortas de 100 caracteres; ajuste si es necesario.

- Las líneas de comentario/*apidoc* deben mantenerse por debajo de 80 columnas.
- Las comas van al final de las líneas ajustadas; los operadores empiezan al comienzo de las nuevas líneas.
- Un operador en el extremo de la línea es fácil perderse si el editor es demasiado estrecho.

3.1.2. Diagrama de componentes

En los diagramas de componentes se muestran los elementos de diseño de un sistema de software. Un diagrama de componentes permite visualizar con más facilidad la estructura general del sistema y el comportamiento de los servicios que estos componentes proporcionan y utilizan a través de las interfaces (25). En la Figura 6 se puede apreciar los diferentes componentes que conforman la solución con las relaciones de uso entre ellos. La implementación de la propuesta de solución está centrada en el componente CIM (por sus siglas en inglés*, Common Information Model*).

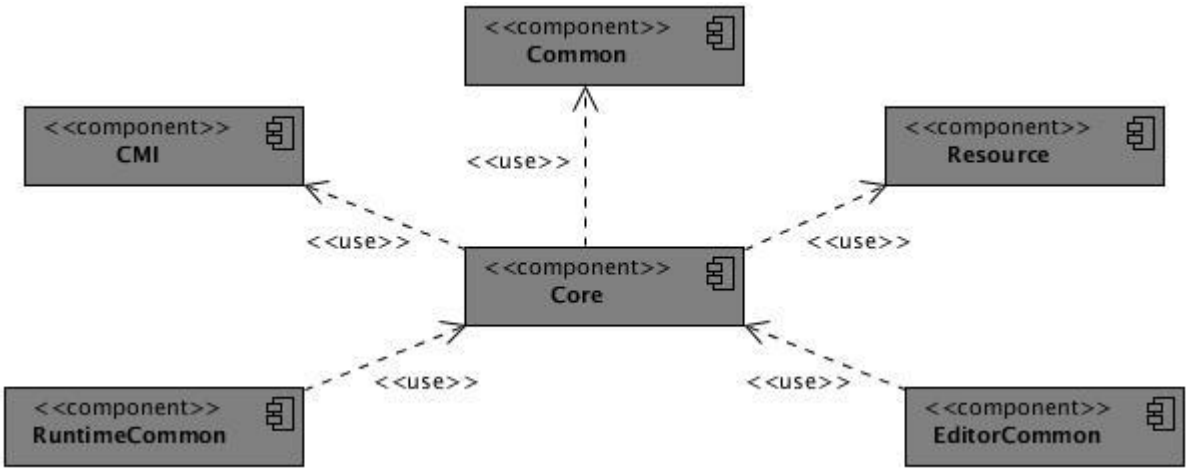


Figura 6. Diagrama de componentes.

3.2. Pruebas

Las pruebas de software son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada. Son un conjunto de situaciones o condiciones ante las cuales un programa debe responder satisfactoriamente para que pueda afirmarse que cumple con sus especificaciones u objetivos. Es una actividad más en el proceso de control de calidad. Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de

dicho proceso de desarrollo. Existen distintos modelos de desarrollo de software, así como modelos de pruebas (38).

3.2.1. Diseño de Casos de Prueba

Los casos de prueba son la forma de verificar las diversas funcionalidades existentes en un producto de software descritas en el formato de los Casos de Uso. Estos se realizan con el objetivo de conseguir una margen de confianza aceptable de que serán encontrados todos los defectos existentes sin consumir una cantidad excesiva de recursos (38).

3.2.2. Validación de la propuesta

Con el objetivo de validar nuestra propuesta de solución, al mismo se le realizaron diversas pruebas una vez ya concluida la etapa de desarrollo con el fin de justificar su valía. Dentro de estos tipos de pruebas se encuentran las pruebas de caja blanca y caja negra, las últimas se llevan a cabo sobre la interfaz del software. Teniendo en cuenta las diversas técnicas que pueden llevarse a cabo durante la aplicación de estas pruebas se decide aplicar la técnica de partición de equivalencia para examinar los valores válidos e inválidos de las entradas existentes en la aplicación.

Tabla 5. Descripción de variable para la selección del formato.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
V1	Formato de proyecto.	Lista de selección	NO	Contiene los valores: NONE, XML, BINARY

Tabla 6. Serializar.

Escenario	Descripción	V1	Respuesta del sistema	Flujo central
EC 1.1 Serializar.	La aplicación	V	Se realiza la	1. Seleccionar las

	debe permitir la serialización de los objetos del proyecto.	XML V BINARY	serialización de los objetos del proyecto.	opciones *Acciones/ Preferencias/ General.* 2. Seleccionar el formato de proyecto. 3. Seleccionar el botón *Guardar*. 4. Seleccionar la opción *Guardar proyecto*.
EC 1.2 Cancelar la selección del formato de proyecto.	La aplicación debe cancelar selección del formato de proyecto.	V XML	Se cancela la selección del formato de proyecto y se mantiene el formato anteriormente seleccionado.	1. Seleccionar las opciones *Acciones/ Preferencias/ General.* 2. Seleccionar o no el formato de proyecto. 3. Seleccionar el botón *Cerrar*.
EC 1.3 No seleccionar el formato para realizar la serialización.	La aplicación no debe permitir la serialización de los objetos del proyecto.	I NONE	Se muestra un mensaje de error y no se realiza la serialización de los objetos del proyecto.	1. Seleccionar las opciones *Acciones/ Preferencias/ General*. 2. Seleccionar el formato de

				proyecto. 3. Seleccionar el botón *Guardar*. 4. Seleccionar la opción *Guardar proyecto*.

Tabla 7. Deserializar.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Deserializar.	La aplicación debe permitir la deserialización de los objetos del proyecto.	Se muestra un interfaz para seleccionar el proyecto. Se realiza la deserialización de los objetos del proyecto.	1. Seleccionar la opción *Acciones.* 2. Hacer clic en el botón *Abrir proyecto.* 3. Seleccionar el proyecto. 4. Seleccionar el botón *Abrir*.
EC 1.2 Cancelar la deserialización.	La aplicación debe cancelar la deserialización de los objetos del proyecto.	Se muestra un interfaz para seleccionar el proyecto. Se cancela la selección del	1. Seleccionar la opción *Acciones.* 2. Hacer clic en el botón *Abrir proyecto.* 3. Seleccionar o no el proyecto.

		proyecto y no se realiza la deserialización.	4. Seleccionar el botón *Cancelar*.
EC 1.3 Seleccionar un proyecto no válido.	La aplicación no debe permitir la deserialización de los objetos del proyecto.	Se muestra un interfaz para seleccionar el proyecto. Se muestra un mensaje de error y no se realiza la deserialización de los objetos del proyecto.	1. Seleccionar la opción *Acciones.* 2. Hacer clic en el botón *Abrir proyecto.* 3. Seleccionar el proyecto. 4. Seleccionar el botón *Abrir*.

Se realizaron 3 iteraciones de pruebas funcionales, de las cuales las 2 primeras resultaron con deficiencias y para la última se erradicaron los defectos encontrados. En la primera iteración se detectaron 5 no conformidades siendo corregidas posteriormente, en la segunda iteración se obtuvieron 3 no conformidades rectificadas de manera inmediata y para una última iteración se arrojaron resultados satisfactorios (ver Figura 7).

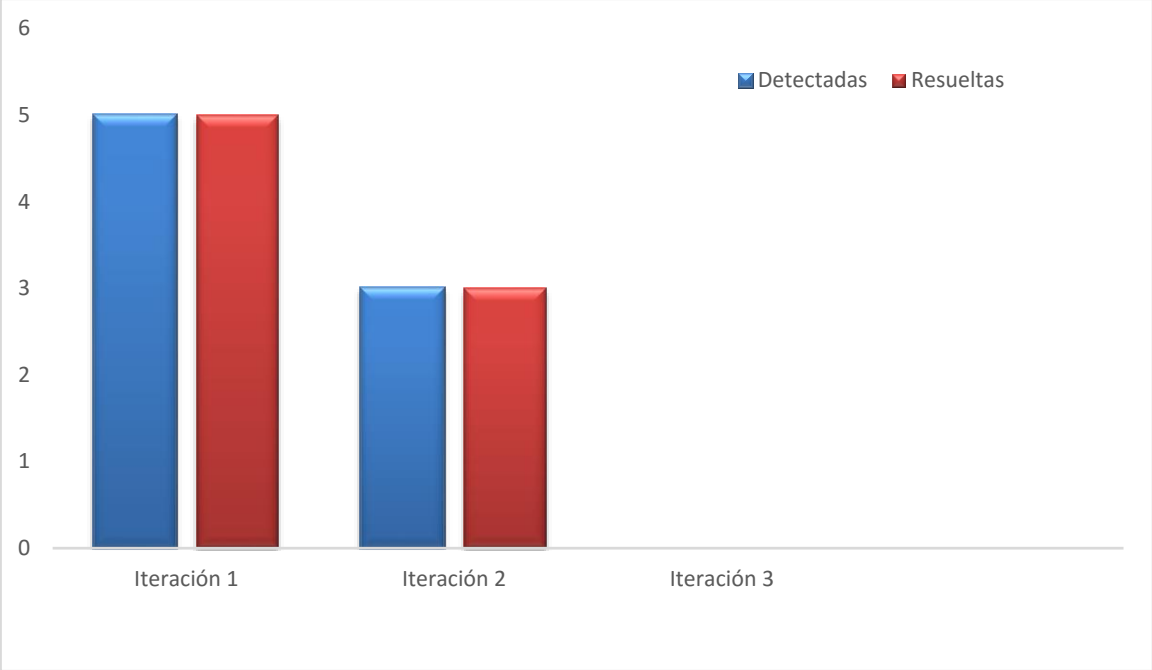


Figura 7: No conformidades por iteración.

No conformidades de la 1ra iteración:

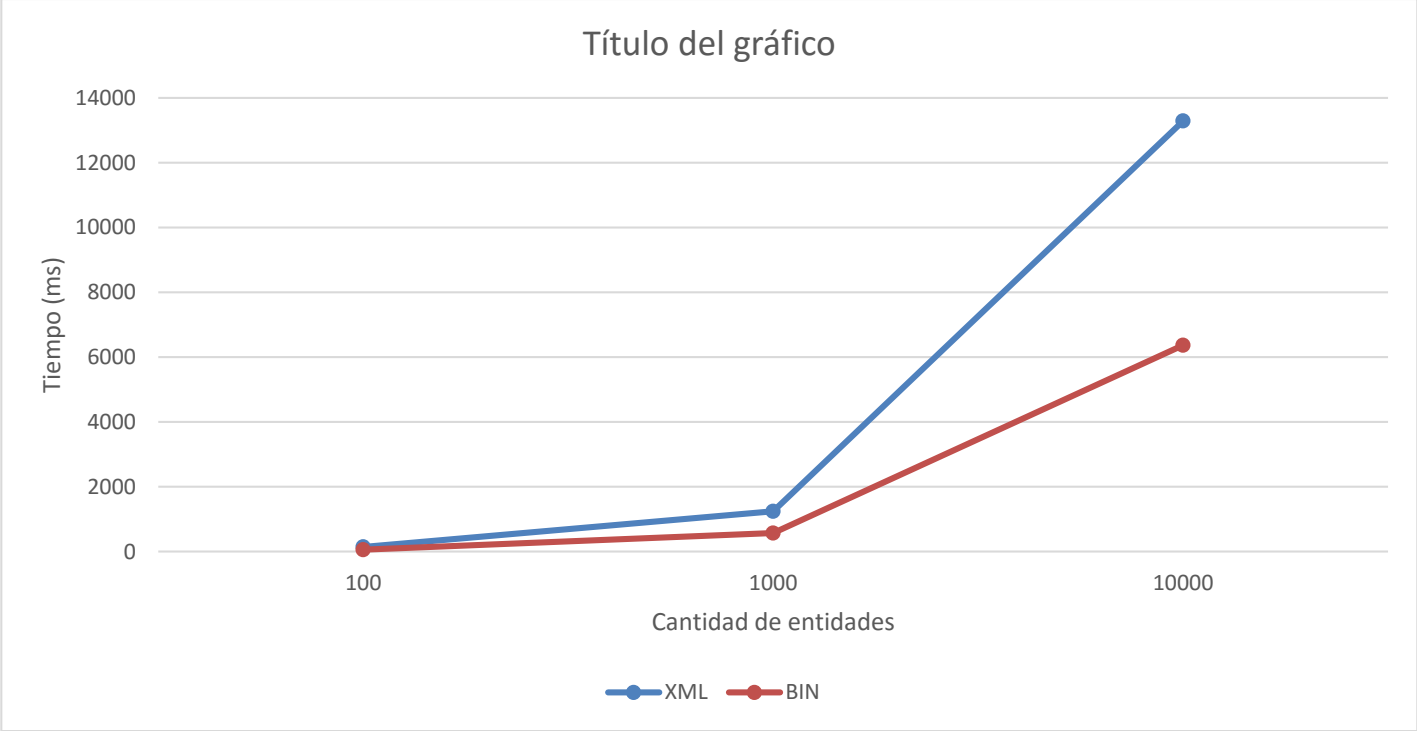
1. Las *propertys* con valor igual a *null* no coincide con la cantidad de *propertys* a serializar (funcionalidad).
2. La clase *ColorForState* serializa/deserializa la descripción y el *QColor* en orden inverso (funcionalidad).
3. Los métodos invocables no reciben los parámetros correctamente (funcionalidad).
4. Error entre los tipos de datos *double* y *float* (funcionalidad).
5. Para las variables de tipo puntero se entra en recursión infinita con las referencias cruzadas (funcionalidad).

No conformidades de la 2da iteración:

1. Cuando no selecciona un formato de proyecto, al dar guardar proyecto, este no se guarda sin notificar al usuario (funcionalidad).
2. Cuando se abre un proyecto que no coincide con el formato de proyecto especificado la herramienta se cierra inesperadamente (funcionalidad).

3. Cuando se selecciona el tipo de formato de proyecto, luego se cierra la aplicación y al volver abrir no tiene el formato especificado.

Se realizaron además pruebas de sistema y de aceptación para evaluar el funcionamiento de la aplicación como un todo y observar el grado de aceptación entre los usuarios finales. A continuación, se muestran los resultados de las pruebas del sistema.



Se probó el software en sistemas operativos de las distribuciones de Debian y Ubuntu de GNU/Linux y se concluye que la propuesta es aceptable, además el uso de la misma debe de ser en computadoras con más de 1gb de RAM y 40gb de disco duro.

3.3. Conclusiones parciales

Tras definir las características de implementación y pruebas de la propuesta de solución se arrojan las siguientes conclusiones:

- ✓ La modelación del sistema permitió obtener la vista estática del mismo a través de los diagramas de componentes y el diagrama de despliegue que indica la situación física de los componentes lógicos desarrollados.
- ✓ Con el objetivo de comprobar el correcto comportamiento de los requerimientos del sistema se realizaron las pruebas para detectar en tiempo defectos que pudiera presentar el sistema; y se validó la solución propuesta a través de las pruebas de sistema y de aceptación las que permitieron evaluar el funcionamiento del sistema.

CONCLUSIONES

En la presente investigación se obtuvo un mecanismo de serialización y deserialización que disminuye el tiempo de ejecución para cargar y guardar la información gestionada en el ambiente de configuración de la Interfaz Hombre Maquina del sistema SCADA SAINUX. Además, se arriban a las siguientes conclusiones:

1. El estudio y la comparación de las bibliotecas existentes de serialización permitió seleccionar la más adecuada para la propuesta de solución.
2. La identificación de los requisitos funcionales permitió desarrollar una solución completa.
3. La solución desarrollada brinda una interfaz genérica de serialización extensible que posibilita agregar nuevos mecanismos de serialización a partir de la reimplementación de la interfaz genérica mencionada.
4. La implementación del mecanismo de serialización binaria permite guardar/abrir proyectos de forma más rápida.

RECOMENDACIONES

Para seguir bajando los tiempos de serialización se recomienda hacer pruebas de concepto con la metodología de serialización intrusiva mencionada en el capítulo 1, este debe considerarse con el arquitecto general del SCADA SAINUX ya que introduciría un mayor acoplamiento entre los módulos del mismo.

Hacer una implementación en JSON del mecanismo de serialización paras las consideraciones a corto/mediano plazo de un HMI web.

REFERENCIAS BIBLIOGRÁFICAS

1. **Penin., A.R.** *Sistemas SCADA.* s.l. : Marcombo, 2012. ISBN 9788426716477.

2. **Telles, Matthew.** *C++ Timesaving Techniques™ For Dummies.* Indianapolis, Indiana : s.n., 2010. 139.

3. **Pecinovský, Rudolf.** *OOP – Learn Object Oriented Thinking and Programming.* Czech Republic : Academic Series, 2013. ISBN 978-80-904661-9-7.

4. **Reddy, Martin.** *API Design for C++.* Burlington, MA 01803, USA : Elsevier, Inc, 2011. ISBN: 978 0 12 385003 4.

5. **Boyer, Stuart A.** *Scada: Supervisory Control And Data Acquisition.* 2015. SBN-13: 978-1936007097.

6. **ISO C++. [En línea] https://isocpp.org/wiki/faq/serialization.**

7. Jiri Soukup, Petr Machácek. *Serialization and Persistent Objects: Turning Data Structures into Efficient Databases.* s.l. : Springer, 2014. ISBN-13: 978-3642393228 .

8. Lott, Steven F. *Mastering Object-oriented Python. Grasp the intricacies of object-oriented programming in Python in order to efficiently build powerful real-world applications.* BIRMINGHAM - MUMBAI : Packt Publishing Ltd, 2014. ISBN 978-1-78328-097-1.

9. Foy, Brian d. *Mastering Perl.* Sebastopol, CA : O'Reilly Media, Inc., 2014. ISBN: 978-1-449-39311-3.

10. Raoul-Gabriel Urma, Mario Fusco, and Alan Mycroft. *Java 8 in Action: Lambdas, streams, and functional-style programming.* s.l. : Manning Publications Co., 2015. ISBN: 9781617291999.

11. *Object Serialization: A Study of Techniques of Implementing Binary Serialization in C++, Java and .NET.* Clarence J M Tauro, N Ganesan, Saumya Mishra, Anupama Bhagwat. s.l. : Researchgate, 2012, Vols. Volume 45– No.6.

12. Ayala, Ramón Montero. *XML Iniciación y referencia .* Aravaca (Madrid) : Universidad Cornplutense de Madrid, 2004.

13. Thelin, Johan. *Foundations of Qt Development.* s.l. : Apress, 2007. ISBN-13 (pbk): 978-1-59059-831-3.

14. MacCaw, Alex. ***JavaScript Web Applications.*** **1005 Gravenstein Highway North, Sebastopol, CA95472. : O'Relly, 2011. ISBN: 978-1-449-30351-8.**

15. Oren Ben-Kiki, Clark Evans, Ingy döt Net. YAML. [En línea] febrero de 2016. ttp://www.yaml.org/spec/1.2/spec.html.

16. C++ Boost. [En línea] http://www.boost.org/doc/libs/1_60_0/libs/serialization/doc/.

17. Code Google. [En línea] https://code.google.com/p/protobuf/.

18. Msgpack Project. [En línea] http://msgpack.org.

19. Summerfield, Mark. ***Advanced Qt Programming.*** **s.l. : Addison-Wesley, 2011. ISBN-13: 978-0-321-63590-7.**

20. Blanchette, Asmin y Summerfield, Mark. ***C++ GUI Programming with Qt 4, Second Edition.*** **s.l. : Prentice Hall, 2008. Print ISBN-13: 978-0-13-235416-5.**

21. S11n. [En línea] http://s11n.net/]..

22. Brian W. Kernighan, Dennis M.Ritchie. ***C Programming Language.*** **s.l. : Practice Hall Software Series.**

23. Linda M. Laird, M. Carol Brennan. ***Software Measurement and Estimation. A Practical Approach.*** **Hoboken, New Jersey. : Wiley & Sons, Inc., 2006. ISBN 0-471-67622-5.**

24. Pressman, Roger S. ***Ingeniería del Software,Un enfoque práctico.***

25. Larman, Graig. ***UML Y PATRONES. INTRODUCCIÓN AL ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS.*** **s.l. : Prentice Hall Hispanoamericana, S.A, 1999. ISBN 970-1 7-0261-1 .**

26. Visual Paradigm. [En línea] 11 de 2015. https://www.visual-paradigm.com/.

27. C++, The Committee : Standard. ***Standard for Programming Language C ++.*** **2013. N3797.**

28. Stroustrup, Bjarne. ***The C++ Programming Language. Fourth Edition.*** **s.l. : Addison-Wesley, 2013. ISBN-13: 978-0-321-56384-2.**

29. Josuttis, Nicolai M. ***The C++ Standard Library. A Tutorial and Reference. Second Edition.*** **s.l. : Addison-Wesley, 2012. ISBN-13: 978-0-321-62321-8.**

30. Qt Creator IDE. [En línea] 10 de 2015. https://www.qt.io/ide/.

31. Mary Beth Chrissis, Mike Konrad, Sandy Shrum. *CMMI® for Development.* s.l. : Addiso-Wesley, 2011. ISBN-13: 978-0-321-71150-2.

32. A Free and Open Source Geographic Information System. [En línea] http://qgis.org/en/site/.

33. Kniberg, Henrik. *Scrum and XP from the Trenches.* s.l. : Wnterprise Software Development Series.

34. Sparx Systems. *Diagrama de Paquete UML 2.* [En línea] [Citado el: 26 de junio de 2016.] http://www.sparxsystems.com.ar/resources/tutorial/uml2_packagediagram.html.

35. Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford. *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002. ISBN: 0-321-12742-0.

36. Qt Wiki. [En línea] https://wiki.qt.io/Qt_Coding_Style.

37. Wilson, Andy Oram and Greg. *Beautiful Code.* s.l. : O'Relly, 2007. ISBN-13: 978-0-596-51004-6.

38. Ali Mili, Fairours Tchier. *Software Testing.* Hoboken, New Jersey : Lawrence Bernstein, 2015.

ANEXOS

Tabla 8. HU2-Serializar "ObjectsCollection".

Historia de usuario	
Número: 2.2	**Nombre del requisito:** Serializar la clase "ObjectsCollection"
Programador: Alvaro	**Iteración:** 1
Prioridad: Alta	**Tiempo estimado:** 24 horas
Riesgo de desarrollo: Baja	**Tiempo real:** 24 horas
Descripción: Permite serializar un objeto de la clase "ObjectsCollection". La clase es una estructura de datos lineales que contiene varios punteros. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	

Tabla 9. HU3-Serializar "ColorCollection".

Historia de usuario	
Número: 2.3	**Nombre del requisito:** Serializar la clase "ColorCollection".
Programador: Alvaro	**Iteración:** 1
Prioridad: Alta	**Tiempo estimado:** 24 horas
Riesgo de desarrollo: Baja	**Tiempo real:** 24 horas
Descripción: Permite serializar un objeto de la clase "ColorCollection". La clase es una estructura de datos lineales que contiene varios colores de la clase *QColor*. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	

Tabla 10. HU4-Serializar "VariantCollection".

Historia de usuario	
Número: 2.4	**Nombre del requisito:** Serializar la clase "VariantCollection".

Programador: Alvaro	**Iteración:** 1
Prioridad: Alta	**Tiempo estimado:** 24 horas
Riesgo de desarrollo: Baja	**Tiempo real:** 24 horas
Descripción: Permite serializar un objeto de la clase "double". La clase es una estructura de datos lineales que contiene varios *QVariant*. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	

Tabla 11. HU6-Deserializar "ColorForState".

Historia de usuario	
Número: 5.1	**Nombre del requisito:** Deserializar la clase "ColorForState"
Programador: Alvaro	**Iteración:** 1
Prioridad: Alta	**Tiempo estimado:** 24 horas
Riesgo de desarrollo: Baja	**Tiempo real:** 24 horas
Descripción: Permite deserializar un objeto de la clase "ColorForState". La clase que define el estado de una válvula en una tubería de petróleo. Está representado por un color y una descripción.	

Tabla 12. HU7-Deserializar "ObjectsCollection".

Historia de usuario	
Número: 5.2	**Nombre del requisito:** Deserializar la clase "ObjectsCollection"
Programador: Alvaro	**Iteración:** 1
Prioridad: Alta	**Tiempo estimado:** 24 horas
Riesgo de desarrollo: Baja	**Tiempo real:** 24 horas
Descripción: Permite deserializar un objeto de la clase "ObjectsCollection". La clase es una estructura de datos lineales que contiene varios punteros. La estructura de datos lineales puede ser una lista, un	

vector, una cola, entre otros.

Tabla 13. HU8-Deserializar "ColorCollection".

Historia de usuario	
Número: 5.3	**Nombre del requisito:** Deserializar la clase "ColorCollection".
Programador: Alvaro	**Iteración:** 1
Prioridad: Alta	**Tiempo estimado:** 24 horas
Riesgo de desarrollo: Baja	**Tiempo real:** 24 horas
Descripción: Permite deserializar un objeto de la clase "ColorCollection". La clase es una estructura de datos lineales que contiene varios colores de la clase *QColor*. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	

Tabla 14. HU9-Deserializar "VariantCollection".

Historia de usuario	
Número: 5.4	**Nombre del requisito:** Deserializar la clase "VariantCollection".
Programador: Alvaro	**Iteración:** 1
Prioridad: Alta	**Tiempo estimado:** 24 horas
Riesgo de desarrollo: Baja	**Tiempo real:** 24 horas
Descripción: Permite deserializar un objeto de la clase "VariantCollection". La clase es una estructura de datos lineales que contiene varios *QVariant*. La estructura de datos lineales puede ser una lista, un vector, una cola, entre otros.	

Tabla 15. RNF- Portabilidad/Adaptabilidad.

Atributo de Calidad	Portabilidad.

Sub-atributos/Sub-características	Adaptabilidad.
Objetivo	Garantizar la ejecución de la aplicación en ambiente *Desktop* con un sistema GNU/Linux instalado.
Origen	Aplicación.
Artefacto	Aplicación.
Entorno	La aplicación está funcionando correctamente.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
1. a Realizar un flujo de trabajo en la aplicación.	
La aplicación debe ejecutarse en arquitecturas de 32 y 64 bits.	La aplicación funciona correctamente en el entorno.
Medida de respuesta	
Disponibilidad de la aplicación.	

Tabla 16. RNF- Portabilidad/Instalabilidad.

Atributo de Calidad	Portabilidad.
Sub-atributos/Sub-características	Instalabilidad.
Objetivo	Garantizar la ejecución de la aplicación.
Origen	Desarrollador.
Artefacto	Aplicación.
Entorno	El sistema operativo cuenta con las dependencias necesarias para la instalación.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
1. a Ejecutar aplicación.	
Para instalar la aplicación se debe tener instalado la librería *Qt* y otras	La aplicación se instala correctamente.

dependencias del *SCADA SAINUX*.	
1. b Compilar aplicación.	
La aplicación debe compilar con las versiones de Qt 4.8.x y Qt 5.x.	La aplicación se compila correctamente.
Medida de respuesta	
Disponibilidad de la aplicación.	

Tabla 17. RNF- Mantenibilidad/Comprobabilidad.

Atributo de Calidad	Mantenibilidad.
Sub-atributos/Sub-características	Comprobabilidad.
Objetivo	Garantizar que la aplicación funcione durante la ejecución de las pruebas funcionales y pruebas de rendimiento.
Origen	Humano.
Artefacto	Aplicación.
Entorno	La aplicación está funcionando correctamente.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
1. a Pruebas funcionales.	
La aplicación debe permitir la ejecución de las pruebas funcionales para encontrar defectos.	La aplicación funciona correctamente antes las pruebas.
1. b Pruebas de rendimiento.	
La aplicación debe permitir la ejecución de las pruebas de rendimiento para encontrar defectos.	
Medida de respuesta	
Disponibilidad de la aplicación.	