Universidad de las Ciencias Informáticas

Facultad 5

Desarrollo de un inspector de propiedades basado en tecnología Qt

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

Autor:

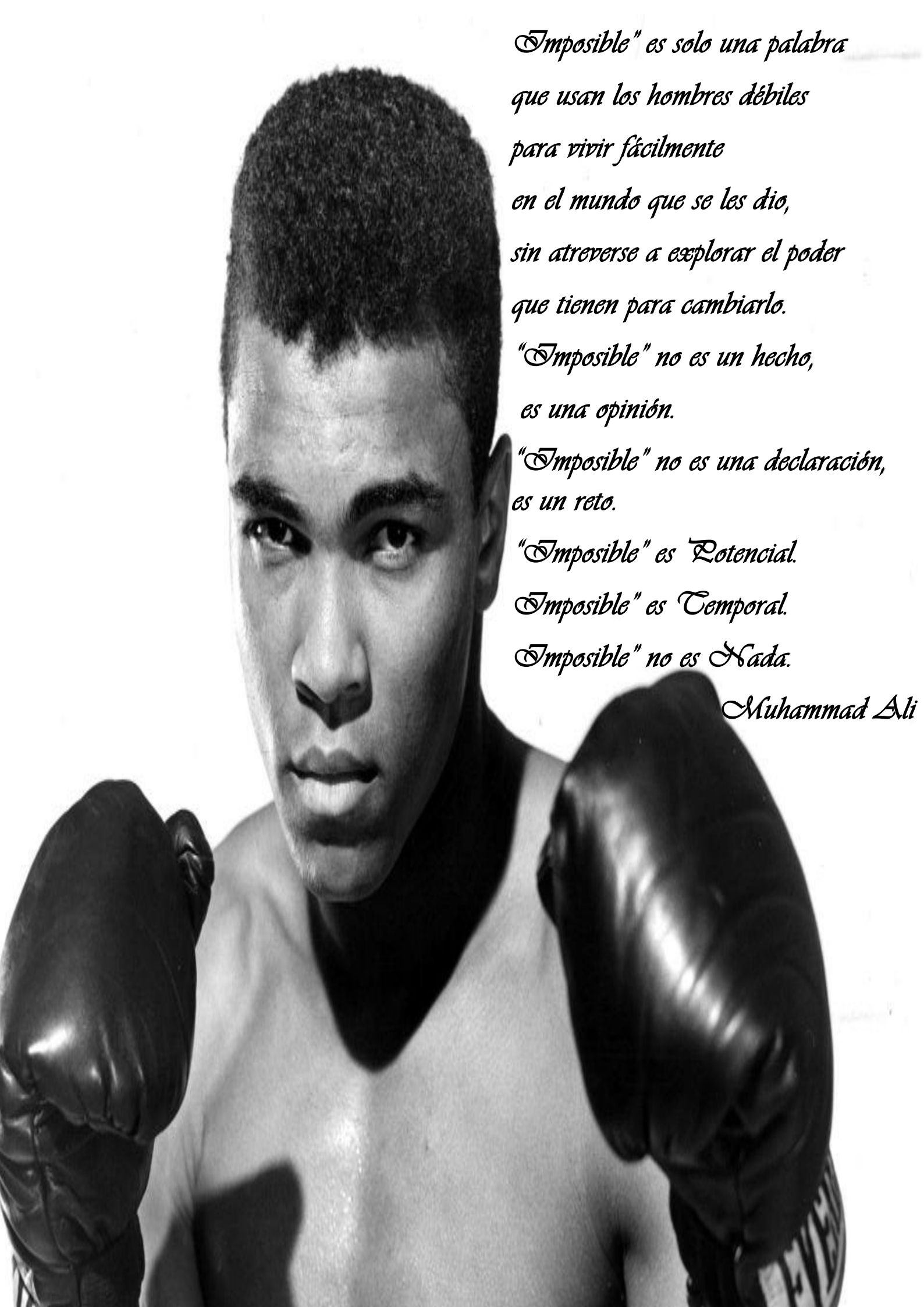
Yohan Diaz Zayas

Tutores:

Ing. Henry Marcelo Cabrera Robles

Ing. José Ernesto Carreño bueno

Ing. Adolfo Yasser Santana Rojas



"Imposible" es solo una palabra
que usan los hombres débiles
para vivir fácilmente
en el mundo que se les dio,
sin atreverse a explorar el poder
que tienen para cambiarlo.
"Imposible" no es un hecho,
es una opinión.
"Imposible" no es una declaración,
es un reto.
"Imposible" es Potencial.
Imposible" es Temporal.
Imposible" no es Nada.

Muhammad Ali

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____

Yohan Diaz Zayas

_____ _____

Ing. Henry Marcelo Cabrera Robles Ing. José Ernesto Carreño Bueno

_____

Ing. Adolfo Yasser Santana Rojas

AGRADECIMIENTOS

De Yohan

Después de muchos años de estudio y esfuerzo, hoy me gradúo de Ingeniero en Ciencias Informáticas. Para hacer realidad este sueño muchas personas me ayudaron, apoyaron y depositaron su confianza en mí. Hoy tengo la oportunidad de hacerles saber cuan agradecido estoy.

A mi mamá Lydia por todo su esfuerzo, sacrificio, por su amor y cariño incondicional, por ser mi mejor amiga(o), por estar conmigo en los momentos más incómodo de mi vida, confiar y ayudarme a realizar este sueño realidad. Quiero que sepa no existe palabra en este mundo que exprese cuanto te quiero y te admiro, por eso y tantas cosas quiero que este logro se tuyo también, aunque nunca podré estar lo suficientemente agradecido por todo lo que ha hecho por mí, gracias por ser mi guía, mi apoyo y mi vida.

A mi papá y abuela por ser la principal motivación e inspiración de este trabajo. Sé que donde estén me están mirando este día. Gracias por tanto amor recibido por ustedes en el tiempo que estuvieron a mi lado.

A mis tías Chichi, Cuca, Tania, Desis por ser mis segunda mamá, gracias por tanto amor y cariño recibido, por estar siempre para mí. Mi tío Jorgito por ser un amigo a su forma, gracias a ti también. Yanet y Gallega por ser como mis tías también gracias.

A mis primos Yoandry, Leycester, Odaismis, Odaymaralis, Yaima, por su cariño, por ser como mis hermanos, gracias a ustedes también. Agradecerle a la vida por darme la posibilidad de tenerlo a todos ustedes y ahora a mi hermana Dainelis, los quiero con la vida.

A mis tutores Adolfo, Henry, José Ernesto por su esfuerzo y paciencia, son una de las piezas más importantes en esta tesis y más que tutores y profesores son amigos para mí y quiero que sepan que pueden contar conmigo para lo que sea.

Durante el transcurso de vida siempre hay personas que dejan huellas en uno y estas personas son los amigos la otra familia, sin duda sé que tengo en ellos buenos hermanos Anniel, Hayron, Armando, Alejandro Suarez, Isleidis, Handy, Vicente (El Viper) gracias a ustedes por estar ahí y soportarme. A mi familia de aquí de la UCI que me llevo en corazón Yamir (El Droga), Rey, Edward, Raidel (El Miki), Yalbert la real familia José Raúl, Orlando (El Hobbit), Eliades (El berraco), Eduardo (El Maestro), Yaikel, Frank (El loco). Los todos los amigos que tenido la posibilidad de hacer y conocer. Los amigos del aula Frank (Tito), Juan Carlos, David, Andy, Manolo y a todos los demás integrantes del

grupo. A Ricardo (Riki), Yoan Parra, Elizabet, Lizandra (la pura). Los amigos del edición Alik, Alexis, Cama, Yaser, El Pop, Lachy, todos los integrantes de mi casa y a todos los amigos del edificio 91 y antiguo 87 gracias a todos ustedes también.

A todos aquellos que de una forma u otra han formado parte de mi vida y me han apoyado en todo momento, Luisito, Diomne, Dayana, Lazaro, Anibal, Jiubel al piquete del futbol, quiero que sepan que pueden contar conmigo para lo que sea que aquí tienen un amigo más. En fin a todos aquellos que de una forma u otra han formado parte de mi vida y me han apoyado gracias a todos. Por siempre **Visca Barca.**

DEDICATORIA

De Yohan

A mi familia, en especial a mi mamá que siempre me ha apoyado y confiando en mí. Mi papá y abuela aunque no se encuentren físicamente sé que donde están me cuidan y me protegen, ustedes son la principal motivación e inspiración de este trabajo, por tanto amor y cariño recibido quiero regalarles este momento y honrarlos. Los quiero mucho.

RESUMEN

El inminente avance de las ciencias informáticas en el desarrollo industrial y económico, ha provocado el crecimiento de la industria cubana del *software*. La Universidad de las Ciencias Informáticas promueve el desarrollo de la informática mediante la realización de proyectos de *software* pertenecientes a los centros de desarrollo, ejemplo de ello es el Centro de Informática Industrial de la Facultad 5.

El uso del inspector de propiedades ha aumentado considerablemente en los proyectos del centro, debido a su vital importancia en el trabajo con entidades u objetos, posibilita la visualización y edición de las características de los objetos. El centro de informática industrial se encuentra en la migración a la versión 5 de Qt, la cual se encuentra retrasada por las limitaciones que presenta el inspector usado actualmente, por lo que se hace necesario el desarrollo de un componente que responda a los intereses del centro.

En el presente trabajo se hace un estudio de algunos de los principales aspectos de los inspectores de propiedades en cuanto a las formas de visualización y funcionamiento. Además se desarrolla un componente, donde se utilizan las técnicas, fases y actividades definidas en la metodología utilizada. Posteriormente se realizan pruebas de aceptación para validar la calidad de solución propuesta.

Palabras clave:

Edición, inspector de propiedades, visualización

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad existe un elevado crecimiento en la demanda de los productos de *software* y los servicios de información tecnológica. Las Tecnologías de la Información y las Comunicaciones (TICs) juegan un papel relevante en la economía mundial. Las TICs forman parte de la mayoría de los sectores: educación, robótica, administración pública, empleo y empresas, salud, entre otras.

La Informática Industrial se enfoca en la aplicación de métodos y técnicas de las ciencias informáticas a los distintos ámbitos de la industria, mediante el tratamiento automático de la información proveniente de los procesos industriales, mediante el uso de ordenadores o computadoras que hacen más eficiente y precisa la toma de decisiones. Dentro de esta área han proliferado los llamados Sistemas de Supervisión, Control y Adquisición de Datos (SCADA, por sus siglas en inglés), los cuales son aplicaciones de *software* diseñadas especialmente para supervisar y controlar a distancia una instalación, proceso o sistema de características variadas, proporciona la comunicación con los dispositivos de campo (sensores, actuadores, controladores) para controlar el proceso desde la pantalla del ordenador. En el desarrollo de estos sistemas, así como para el resto de los productos de *software* es importante prestarle atención a la información que contenga la aplicación, de modo que sea entendible y aprovechable para los usuarios.

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 la Universidad de las Ciencias Informáticas (UCI), se encuentra la Facultad 5, en la cual radica el Centro de Informática Industrial (CEDIN) que tiene como misión la creación de *software* para informatizar los procesos industriales. Para llevar a cabo dicha tarea el CEDIN desarrolla tecnología basada en componentes. Entre los componentes más usados en los proyectos se encuentra el Inspector de Propiedades, que es utilizado para la edición de las propiedades de las entidades, está integrado al módulo interfaz hombre-máquina (HMI, por sus siglas en inglés) del SCADA SAINUX y al Sistema de análisis e interpretación petrofísica (AnPer).

HMI y SCADA son dos términos que están íntimamente relacionados en la medida en que un HMI forma parte de los componentes de un sistema SCADA. HMI, es un panel de control diseñado para conseguir una comunicación interactiva entre operador y

proceso/máquina, con la función de transmitir órdenes, visualizar gráficamente los resultados y obtener una situación del proceso/máquina en tiempo real.

La representación de la información de las entidades en el inspector de propiedades QtPropertyBrowser es uno de los inconvenientes a los que se enfrentan los especialistas en este tipo de aplicaciones, dado que el sistema no visualiza correctamente algunas propiedades del objeto en inspección, además presenta otras dificultades, las cuales son:

- No presentan una correcta organización de los elementos básicos de comunicación entre las interfaces de usuarios, esto trae como consecuencia que cuando existe una gran cantidad de propiedades se hace difícil su búsqueda.
- No es desarrollado por la UCI lo que trae consigo la necesidad de desarrollar uno que se acoja a las necesidades del centro.
- No tiene soporte para el *framework* Qt en su versión 5 por lo que dificulta la migración de las aplicaciones.
- Presenta latencia para los objetos de grandes cantidades de propiedades, lo que conlleva a la pérdida de tiempo en el proceso y a la inconformidad de los usuarios.

Debido a la situación existente se plantea como **problema de la investigación** a resolver:
¿Cómo apoyar a la gestión de propiedades de las entidades que se utilizan en las soluciones del CEDIN con tecnología Qt?

Teniendo en cuenta el problema anterior se determina como **objeto de estudio:** Gestión de las propiedades de las entidades con tecnología Qt.

Enmarcado en el **campo de acción:** Gestión de propiedades de las entidades con tecnología Qt en las soluciones del CEDIN.

Se plantea como **objetivo general:**

Desarrollar un componente para la gestión de propiedades de las entidades que se pueda integrar a las soluciones del CEDIN con tecnología Qt.

Para dar solución al objetivo general antes propuesto se definieron las siguientes **tareas de la investigación:**

- Elaboración del marco teórico conceptual a partir del estado del arte referente al tema.
- Investigación de sistemas con inspectores de propiedades.

- Estudio y selección de los patrones de diseño y arquitectura para su utilización en el diseño de la interfaz de usuario y la representación de los diagramas de clase que van a guiar el proceso de implementación de la solución.
- Diseño e implementación de prueba de concepto del componente inspector de propiedades.
- Implementación del componente inspector de propiedades funcional.

El **posible resultado** es un componente de inspección de propiedades y la especificación de requerimientos del sistema.

Métodos científicos utilizados en la investigación

Métodos teóricos

- **Análisis Histórico-Lógico:** Este método se utiliza para realizar el análisis del estado del arte relacionado con los inspectores de propiedades, la causa que origina el problema, sus principales conceptos, ventajas y desventajas, elementos que definen una buena organización y estructuración de la información.
- **Analítico-Sintético:** Es utilizado para realizar un análisis integral de los elementos y fundamentos teóricos principales que integran los inspectores de propiedades.

Métodos empíricos

- **Entrevista:** Este método fue aplicado para validar los datos necesarios para la solución del problema de la investigación y para la obtención del conocimiento necesario para dar solución al problema planteado.

El presente trabajo está conformado de la siguiente manera:

Capítulo 1. Fundamentación teórica: En este capítulo se abordan definiciones y conceptos importantes de los inspectores de propiedades, SCADA y HMI. Se especifican las características, ventajas y desventajas de las técnicas y herramientas que son utilizadas para dar solución al problema expuesto en la presente investigación.

Capítulo 2. Análisis y diseño de la solución: En este capítulo se detalla la propuesta de solución del componente a través de la metodología escogida. Además, se obtiene a partir del diseño, la arquitectura base, la descripción de las principales clases que fueron definidas, los diagramas de clases y se hace referencia a los principales patrones de diseño que fueron utilizados.

Capítulo 3. Implementación y prueba: En este capítulo se muestran los aspectos vinculados con la implementación de un inspector de propiedades, así como los estándares de codificación que permiten crear un código fácil de entender. Las tareas de ingenierías realizadas para cumplir con las historias de usuario y se finaliza con la realización del proceso de pruebas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se abordará acerca del estado del arte, tendencias actuales y los principales elementos que se tienen en cuenta para el desarrollo de un inspector de propiedades, además de los conceptos y definiciones de diferentes autores. También contiene las tecnologías y herramientas seleccionadas para realizar el diseño e implementación de la solución, así como la metodología usada.

1.1 Conceptos asociados a este trabajo

1.1.1 Inspector

El uso de inspectores es muy común en la actualidad debido a las ventajas que provee. Según el diccionario de la Real Academia Española, inspeccionar es examinar, reconocer atentamente. Enfocándose en el área de la ciencia, un inspector no es más que una forma para examinar un objeto para adquirir de este información que es útil.

1.1.2 Propiedad

Una propiedad es un atributo de un objeto que define una de las características del objeto, como tamaño, color o ubicación en pantalla, o un aspecto de su comportamiento, por ejemplo, si está habilitado o visible. Para cambiar las características de un objeto, se cambian los valores de las propiedades correspondientes (Developer Network, 2015).En la Figura 1 se muestra un inspector de propiedades.

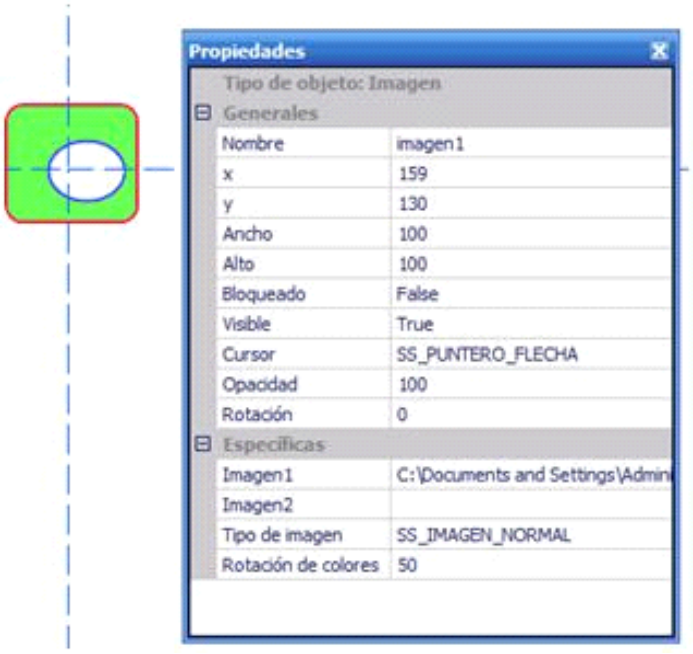


Figura 1. Algunas de las propiedades de un objeto.

1.2 Aplicaciones que utilizan inspectores de propiedades

Son muchas las aplicaciones que utilizan inspectores de propiedades para facilitar el trabajo con objetos. Los inspectores pueden variar en cuanto a su forma de visualizar las propiedades, las cuales pueden ser de forma arborescente, acordeón, pestaña, entre otras. A continuación se hace una breve descripción de diferentes inspectores de propiedades.

1.2.1 Inspector de propiedades Adobe Flash

El inspector facilita el acceso a los atributos más utilizados de la selección realizada, ya sea en el escenario o en la línea de tiempo. Se pueden modificar los atributos del objeto o el documento en el examinador de propiedades sin acceder a los menús o paneles que contienen estos atributos.

Este muestra información y la configuración del elemento que está seleccionado, que puede ser un documento, un texto, un símbolo, una forma, un mapa de bits, un vídeo, un grupo, un fotograma o una herramienta. Cuando hay dos o más tipos de objetos seleccionados, el inspector de propiedades muestra el número total de objetos seleccionados (Ayala p. 2015). En la Figura 2 se muestra un inspector de propiedades de Flash.

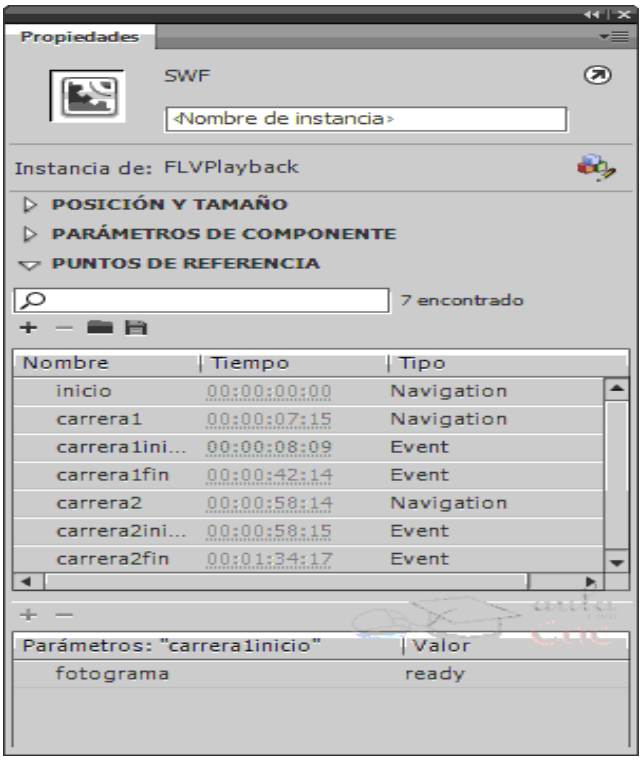


Figura 2. Inspector de propiedades de Flash.

1.2.2 Inspector de propiedades Unity 3D

Los juegos en Unity 3D están compuestos de múltiples GameObjects que contienen mallas, guiones, sonidos, u otro elemento gráfico como luces. La clase Inspector muestra información detallada sobre el GameObject seleccionado y todos los componentes adjuntos y sus propiedades. Cualquier propiedad que se muestre en el inspector puede ser directamente modificado (Huanay Martínez, 2010).

La ventana del examinador ofrece la información disponible del objeto seleccionado actualmente en la ventana de jerarquía o en la ventana del proyecto. En ella se visualizan los componentes actuales del objeto seleccionado así como todas sus propiedades. Ofrece la posibilidad de cambiar y editar la mayoría de los campos visibles de dichos objetos, con un efecto inmediato en el resto de las ventanas (Aris Goicoechea Lassaletta, 2012). . En la Figura 3 se muestra un inspector de propiedades de Unity 3D.

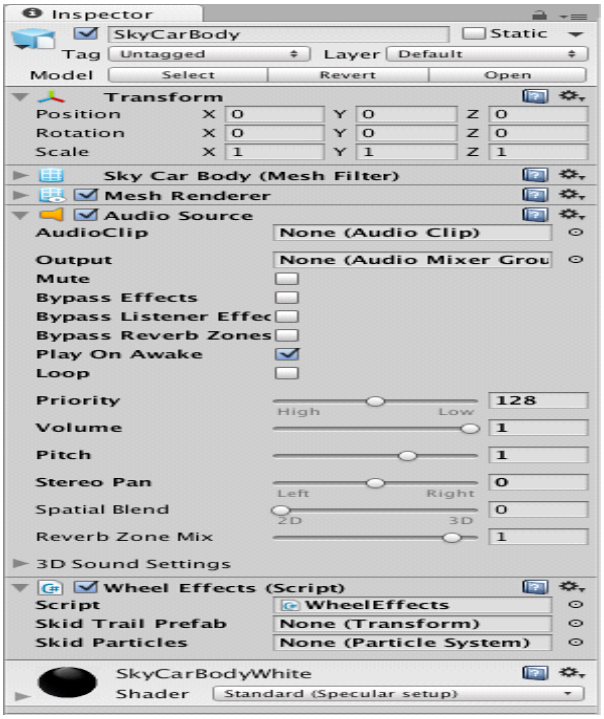


Figura 3. Inspector de propiedades de Unity 3D.

1.2.3 Inspector de propiedades Adobe Dreamweaver

El inspector permite examinar y editar las propiedades del elemento que esté seleccionado, puede ser un objeto o texto. Es posible seleccionar los elementos en la ventana de documento o en el inspector de código.

La mayoría de los cambios realizados en las propiedades se aplican de inmediato en la ventana de documento. (Para algunas propiedades, los cambios no se aplican hasta

que se hace clic fuera de los campos de texto de edición de la propiedad, presiona la tecla Enter o Tab para cambiar a otra propiedad.)

El contenido del mismo varía en función del elemento seleccionado. Si se quiere obtener información sobre propiedades concretas, se selecciona un elemento en la ventana de documento y, a continuación, se hace clic en el icono Ayuda, situado en la esquina superior derecha del inspector.

Este muestra inicialmente las propiedades del elemento seleccionado que se utiliza con mayor frecuencia; para ver todas las propiedades se debe hacer clic en la flecha de ampliación situada en la esquina inferior derecha del inspector de propiedades, para ver más propiedades del elemento (Castaño, 2007). . En la Figura 4 se muestra un inspector de propiedades de Dreamweaver.

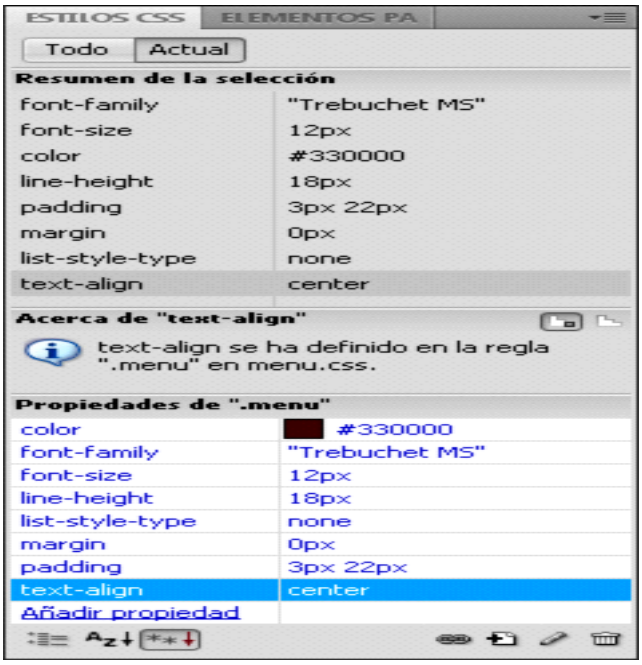


Figura 4. Inspector de propiedades de Dreamweaver.

1.2.4 Inspector de propiedades del HMI del CEDIN.

En el CEDIN, son muchos los componentes que se utilizan para llevar a cabo las misiones, pero entre los componentes más usados en los proyectos se encuentra el inspector de propiedades que sirve para la edición de las propiedades de las entidades u objetos, además:

- Permite la inspección de las propiedades de los objetos, facilitando así el trabajo.
- Permite el filtrado de propiedades, permitiéndole a los usuarios un trabajo más ágil y cómodo.

- Permite la agrupación de propiedades, distinguiéndose cada grupo con un color diferente.
- Soporta los diferentes tipos de datos del *framework* Qt.

En la Figura 5 se muestra el inspector de propiedades del HMI del CEDIN.

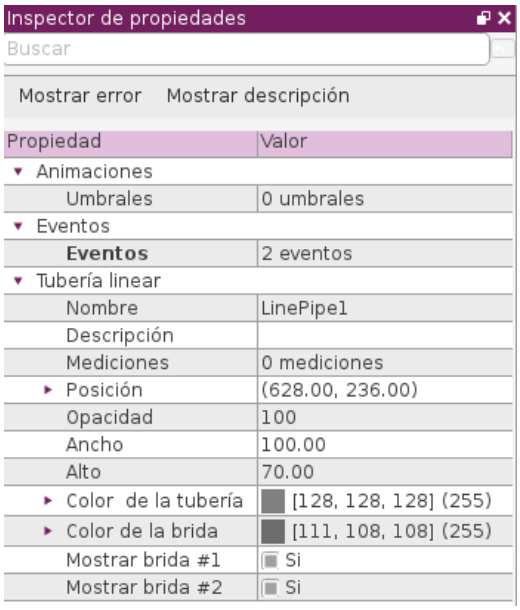


Figura 5. Inspector de Propiedades del HMI del CEDIN.

1.3 Funcionalidades que proveen los inspectores de propiedades.

- Soporte parcial para ASDF, el sistema de Descubrimiento de Servicios Desarrollado para el Middleware de Hesperia. Permite observar en vivo el proceso de anunciamiento de objetos y servicios, comprobar la validez de las referencias remotas suministradas y acceder directamente a la interfaz y funcionalidad del servicio anunciado.
- Inspección de cualquier objeto que utilice las interfaces elementales para sensores y actuadores de todo tipo.
- Soporte completo para objetos activos. Permite a la interfaz gráfica actualizar automáticamente el valor de cualquier variable representada por un objeto remoto capaz de manipular su propio canal de eventos.
- Inspección de objetos compuestos y elementos de contexto. Ofrece la posibilidad de listar y modificar (añadir/eliminar) elementos de agrupaciones de objetos.

1.4 Sistemas de Supervisión, Control y Adquisición de Datos.

Los SCADA, son aplicaciones de *software*, diseñadas con la finalidad de controlar y supervisar procesos a distancia. Se basan en la adquisición de datos de los procesos

remotos. Son una aplicación de software diseñada para funcionar sobre ordenadores en el control de la producción, posibilitar la comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, entre otros) y controlar el proceso de forma automática desde la pantalla del ordenador. Además, envía la información generada en el proceso productivo a diversos usuarios, tanto del mismo nivel como hacia otros supervisores dentro de la empresa, es decir, que permite la participación de otras áreas como control de calidad, supervisión, mantenimiento y otros (García p. 2006). En la Figura 6 se muestra un sistema de SCADA.

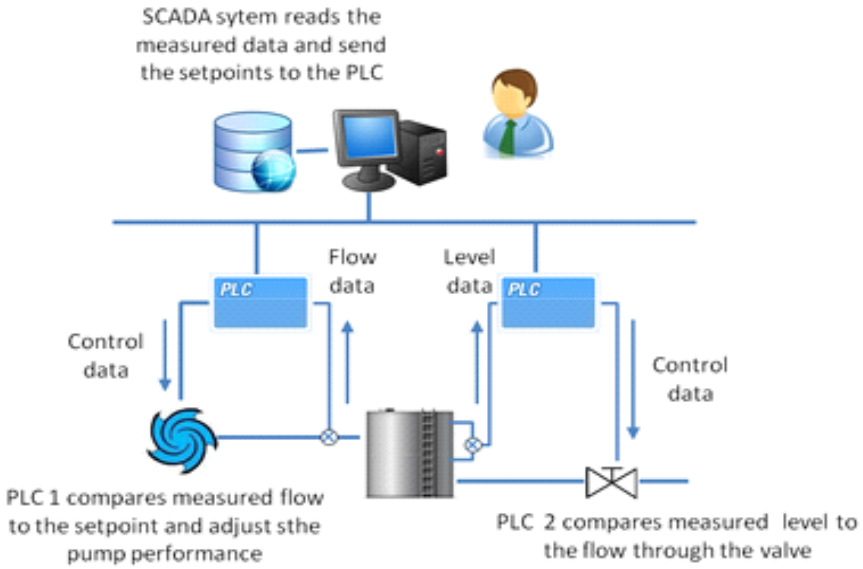


Figura 6. Sistema de SCADA.

1.4.1 Funciones generales.

- Adquisición de datos, para recoger, procesar y almacenar la información recibida.
- Supervisión, para observar desde un monitor la evolución de las variables de control.
- Control, para modificar la evolución del proceso, actuar sobre los reguladores autónomos básicos (consignas, alarmas, menús) directamente sobre el proceso mediante las salidas conectadas.

1.4.2 Funciones más específicas.

- Transmisión de información con dispositivos de campo y otras computadoras.
- Base de datos: Gestión de datos con bajos tiempo de acceso. Suele utilizar ODBC, Conectividad abierta de Bases de Datos.
- Presentación: Representación gráfica de los datos. Interfaz de Operador o HMI.
- Explotación de los datos adquiridos para la gestión de la calidad, control estadístico, gestión de la producción y gestión administrativa y financiera.

1.5 Sistema SCADA SAINUX

Permitir la comunicación entre los distintos módulos que componen un SCADA resulta muy engorroso debido a que no siempre deben estar soportados por la misma plataforma, el mismo sistema operativo o el mismo lenguaje de programación, por lo que resulta de vital importancia contar con un medio de comunicación que resuelva este tipo de problemas. Además, a partir de las limitantes que tiene Cuba para el pago de licencias de *software*, se hace necesario el desarrollo de productos informáticos basados en tecnologías libres y que a la vez permitan obtener una versión comercial del producto final. Esta tecnología está liberada por la Licencia Pública General Menor (LGPL) que permite obtener un producto final sin la obligación de redistribuir cualquier parte del código, apoya así el progreso de la economía nacional que puede proporcionar el mercadeo de *software* (Lalcebo, 2013).

1.6 Módulo Interfaz Hombre-Máquina

HMI, es el dispositivo o sistema que permite el interfaz entre la persona y la máquina. Los sistemas HMI se pueden pensar como una "ventana" de un proceso. Esta ventana puede estar en dispositivos especiales como paneles de operador o en un computador. Los sistemas HMI en computadores se les conoce como software HMI (en adelante HMI) o de monitorización y control de supervisión. Las señales del proceso son conducidas al HMI por medio de dispositivos como tarjetas de entrada/salida en el computador, PLC´s (Controladores lógicos programables), RTU (Unidades remotas de I/O) o DRIVE´s (Variadores de velocidad de motores). Todos estos dispositivos deben tener una comunicación que entienda el HMI (JOSÉ, 2012). En la Figura 7 se muestra una Interfaz Hombre Máquina.

Figura 7. Interfaz Hombre Máquina (HMI).

1.6.1 Ventajas de los sistemas SCADA/HMI

- Potenciación del control y monitorización de su proceso, sistema o planta.

- Brinda información importante como avisos, alarmas que a tiempo prevendrán de paradas de la planta.
- Incremento de la productividad al disminuir las paradas del proceso industrial.
- Simplificación de la interacción de diferentes sistemas, de distintos fabricantes, con múltiples comunicaciones.
- Optimización de la efectividad del usuario al controlar más datos, y ayudar así con alarmas o agrupaciones de datos a conocer el estado del sistema y sus problemas.
- Aumento de la calidad por el buen funcionamiento del sistema completo.
- Reducción de los costes operacionales, tanto de integración como de mantenimiento (JOSÉ, 2012).

1.7 Metodología de desarrollo de *software*

Las metodologías de desarrollo de *software* se utilizan para estructurar, controlar y planificar el proceso en sistemas de información. En la actualidad existen varias propuestas, entre las cuales se encuentran las tradicionales que se basan específicamente en el control de proyectos de gran tamaño. Sin embargo, a medida que pasan los años se ha demostrado que las metodologías tradicionales no ofrecen una correcta solución a proyectos donde los requisitos no se conocen con exactitud. Debido a esta dificultad presentada surgen las metodologías ágiles las cuales intentan adaptarse a la realidad del *software* (Hérnandez Muñoz, 2006).

La UCI cuenta con centros productivos, en su gran mayoría pertenecientes a las facultades que conforman a la Universidad. Cada uno de estos centros se dedica al desarrollo de *software* y/o servicios asociados. Todos los desarrollos se encuentran organizados en proyectos, clasificados en: Gestión, Componentes, tecnología base, Portales y Sistemas operativos. Esta diversidad de centros y proyectos trae consigo la heterogeneidad en el proceso de desarrollo de *software* (Sánchez, 2014).

Por lo tanto, la universidad decide adaptar una metodología para establecerla como estándar en todos sus centros productivos. La metodología escogida fue el Proceso Unificado Ágil (AUP) que es una versión simplificada del Proceso Unificado de Desarrollo (RUP). Esta describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de *software* de negocio con técnicas ágiles y conceptos que aún se mantienen válidos en RUP. Para adaptar esta metodología a las necesidades de la universidad se le realizaron variaciones y se confeccionó la metodología AUP-UCI, que es la utilizada durante el transcurso de esta investigación.

Entre las técnicas ágiles que utiliza AUP se encuentra el modelado ágil, se hará uso de esta técnica para los proyectos que necesiten por sus características encapsular sus requisitos funcionales en Historias de usuario o en descripción de requisitos por procesos. La otra forma de encapsular los requisitos se mantiene por Casos de Uso (Sanchez, 2014). A continuación, se muestran las fases y las disciplinas de esta variación de la metodología AUP.

1.7.1 Fases AUP-UCI

Fases	Objetivos [23]
Inicio	• Llevar a cabo las actividades relacionadas con la planeación del proyecto. • Realizar un estudio inicial de la organización cliente. • Realizar estimaciones de tiempo, esfuerzo y costo.
Ejecución	• Ejecutar las actividades requeridas para desarrollar el software. • Ajustar los planes del proyecto considerando los requisitos y la arquitectura. • Modelado del negocio. • Obtener los requisitos. • Elaborar la arquitectura y el diseño. • Implementar y liberar el producto. • Transferir el producto al cliente. • Capacitar a los usuarios finales sobre la utilización del software.
Cierre	• Analizar los resultados del proyecto y su ejecución • Realizan las actividades formales de cierre del proyecto.

Tabla 1. Fases AUP-UCI.

Disciplinas	Objetivos [23]
Modelado de negocio (opcional)	Comprender los procesos de negocio
Requisitos	Administrar y Gestionar los requisitos funcionales y no funcionales.
Análisis y diseño	Modelar el sistema.
Implementación	Construir el sistema.
Pruebas interna	Verificar el resultado de la implementación.
Pruebas de liberación	Diseñar y Ejecutar pruebas por una entidad Externa Certificadora de la calidad.
Pruebas de Aceptación	Verificar que el software está listo.
Despliegue (Opcional)	Instalar, Configurar, Adecuar, y Poner en marcha.
Gestión y soporte	-

Tabla 2. Disciplinas AUP-UCI.

1.7.2 Ventajas de AUP

- El personal sabe lo que está haciendo: no obliga a conocer detalles.
- Simplicidad: apuntes concisos.
- Agilidad: procesos simplificados del RUP.
- Centrarse en actividades de alto valor: esenciales para el desarrollo.
- Herramientas independientes: a disposición del usuario.

1.7.3 Desventajas de AUP

- El AUP es un producto muy pesado en relación al RUP.
- Como es un proceso simplificado, muchos desarrolladores eligen trabajar con el RUP, por tener a disposición más detalles en el proceso (Flores, 2012).

1.8 Ambiente de desarrollo

1.8.1 Qt

Biblioteca multiplataforma utilizada para construir aplicaciones en C++ con interfaces gráficas o no en dependencia de las necesidades del desarrollador. Esta patentado bajo la licencia GPL 2 y LGPL. Actualmente Qt está es propiedad de la empresa finlandesa Nokia quien la adquirió de Trolltech. Nokia utiliza Qt en aplicaciones para dispositivos móviles y aplicaciones de escritorio para computadoras. Actualmente es utilizado por más de 500 000 desarrolladores de todo el mundo y entre las empresas que elaboran productos con esta biblioteca se encuentran Jolla, Navico, ABB, Thales. Entre las

utilidades que brinda esta biblioteca se encuentran las de diseño de interfaces de usuario, dibujar gráficas 2D y 3D de alta calidad utilizadas en diversos ambientes como en la gestión de negocios, herramientas de monitorización entre otras (DIGIA OYJ, 2013).

1.8.2 Qt Creator

Un Entorno de Desarrollo Integrado (IDE) es una aplicación compuesta por un conjunto de herramientas útiles para un programador. Un IDE puede ser exclusivo para un lenguaje de programación o para varios. Suele consistir de un editor de código, un compilador, y un constructor de interfaz gráfica GUI (Alegsa, 2010). A continuación se presenta el IDE seleccionado para dar solución al problema existente.

Qt Creator es un IDE creado por Trolltech para el desarrollo de aplicaciones con las bibliotecas Qt. Está disponible para los sistemas operativos GNU/Linux, Mac OS y Windows, permitiendo al desarrollador crear aplicaciones para múltiples sistemas o plataformas móviles (Albán, 2010). Este IDE se caracteriza por tener un editor de código avanzado con variadas opciones de organización y completamiento de código así como un diseñador de formularios intuitivo y útil. Otras de las funcionalidades que posee es el refactorizado de código.

A continuación se presentan algunas de las características fundamentales de Qt Creator:

- Utiliza el lenguaje de programación orientado a objetos C++.
- Se basa en el *framework* Qt.
- Permite realizar programación visual y programación dirigida por eventos.

1.8.3 Lenguaje de programación C++

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos *hardware* y *software* existentes. Tiene la capacidad de especificar, de forma precisa, cuáles son los datos que debe trabajar un equipo informático, de qué modo deben ser conservados o transferidos dichos datos y qué instrucciones debe poner en marcha la computadora ante ciertas circunstancias.

C++ es un lenguaje imperativo, orientado a objetos, derivado de C. Al igual que C; C++ está muy ligado al *hardware* subyacente, manteniendo una considerable potencia para la programación a bajo nivel; pero se le han añadido elementos que permiten también un estilo de programación con alto nivel de abstracción. Debido a esto; C++ brinda la posibilidad de crear clases, plantillas, sistema de espacios de nombres y funciones en

línea, posee un mecanismo para el manejo de excepciones, permite la sobrecarga de operadores y utiliza operadores para el manejo de memoria. Este lenguaje de programación esta estandarizado por la Organización Internacional de Estándares (ISO) y cuenta con una biblioteca estándar de alta calidad (STROUSTRUP, 1986).

C++ es un lenguaje de propósito general basado en el lenguaje C, al que se han añadido nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, funciones online, sobrecarga de operadores, referencias, operadores para manejo de memoria persistente y algunas utilidades adicionales de biblioteca (Hernán, 2003).

Lenguaje de Modelado Unificado (UML)

El (UML) es un lenguaje de modelado visual que es usado para especificar, visualizar, construir y documentar artefactos de un sistema (JACOBSON p. 2000). Es una técnica de modelado de objetos que persigue abstraer cualquier tipo de sistema, mediante diagramas (de implementación, de comportamiento o interacción, de casos de uso, de clases, entre otros), los cuales son representaciones gráficas que contienen toda la información notable del sistema. Se utiliza para especificar o describir métodos o procesos, es el lenguaje que se usa para describir un modelo.

1.8.5 Visual Paradigm 8.0

Visual Paradigm constituye una herramienta CASE propietaria con licencia gratuita y está diseñada para Análisis y Diseño, utiliza el UML. Tiene disponibilidad para disímiles versiones y para integrarse en múltiples plataformas. Permite que se genere código en varios lenguajes, entre ellos C++.

Se caracteriza por:

- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Licencia gratuita y comercial.
- Disponibilidad en múltiples plataformas. (Schmuller, 2000)

Ventajas:

Apoya todo lo básico en cuanto a artefactos generados en las etapas de definición de requerimientos y de especificación de componentes.

- Tiene apoyo adicional en cuanto a generación de artefactos automáticamente.
- Disponibilidad en múltiples plataformas: Microsoft Windows (98, 2000, XP, Vista o superior), Linux, Mac OS X, Solaris o Java.
- Brinda la posibilidad de intercambiar información mediante la importación y exportación de ficheros con aplicaciones como por ejemplo Visio y Rational Rose.
- Generación de código e ingeniería inversa: genera el código a partir de los diagramas, para las plataformas como .Net, Java y PHP, además obtiene los diagramas a partir del código.
- Generación de documentación: permite documentar todo el trabajo sin necesidad de utilizar herramientas externas.

Desventajas:

- Las imágenes y reportes generados, no son de muy buena calidad. Instalación costosa.
- Los modelos a veces no pueden ser reabiertos.
- No hay llamadas reflexivas en los diagramas de secuencia.
- Se debe seleccionar una clase para crear un diagrama de secuencia.

1.9 Conclusiones parciales

En este capítulo se realizó una investigación sobre los inspectores de propiedades, además se abordó sobre los sistemas SCADA, donde se trató el SCADA SAINUX y la interfaz HMI. Por último se definieron las herramientas y tecnologías a utilizar, como metodología de desarrollo de *software* se eligió AUP-UCI, como IDE Qt Creator en su versión 3.0, como lenguaje de programación C++ y para el modelado la herramienta case Visual Paradigm en su versión 8.0.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA SOLUCIÓN

Introducción

En el siguiente capítulo se muestra la descripción de la propuesta de solución. Primeramente se describe cada uno de los procesos de negocio mediante los cuales se capturan los requisitos necesarios para la realización del sistema, se enumera y especifican cada uno de ellos, tanto los funcionales como los no funcionales. Se muestran los artefactos correspondientes al diseño de clases y modelo de datos.

2.1 Análisis y diseño

2.1.1 Modelo de dominio

El modelo de dominio es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Este es de gran ayuda para desarrolladores y usuarios, debido a que utilizan un vocabulario común y pueden entender el contexto en que se enmarca el sistema (Sánchez, 2016). Representa clases conceptuales del dominio del problema, conceptos del mundo real en lugar de componentes de *software*. En la Figura 7 se muestra el modelo de dominio

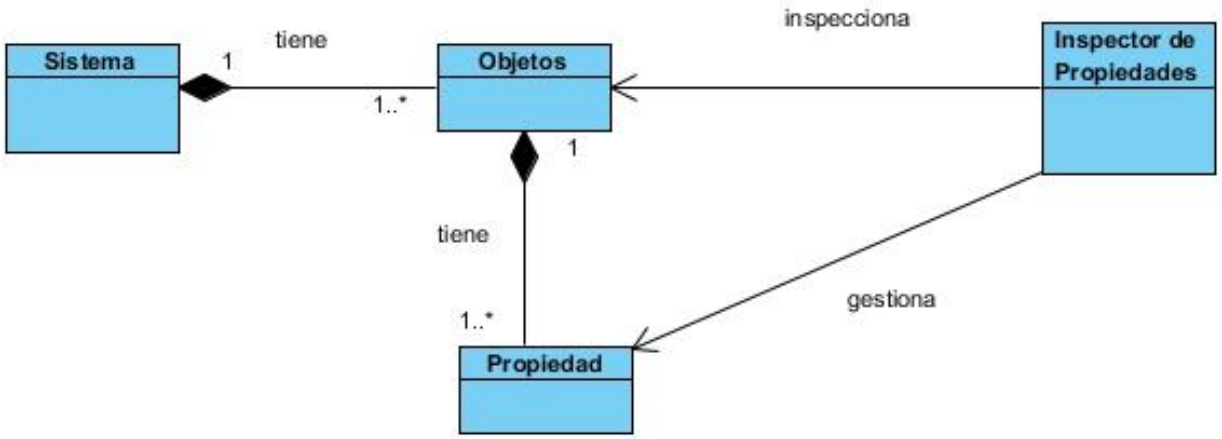


Figura 8. Modelo de dominio.

El sistema tiene diversos objetos y cada objeto es una instancia de clase, que contiene propiedades referentes a dicho objeto, estas propiedades son gestionadas por el inspector de propiedades, mientras este inspecciona a dicho objeto.

2.2 Diagrama de paquetes

El diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas y evidencian sus dependencias. Los paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada uno y minimizar el acoplamiento externo entre ellos (Gutiérrez, 2009). En este trabajo el diagrama muestra la relación

entre los paquetes que presenta el inspector de propiedades para mejorar su comprensión y entendimiento. En la Figura 8 se muestra el diagrama de paquetes.

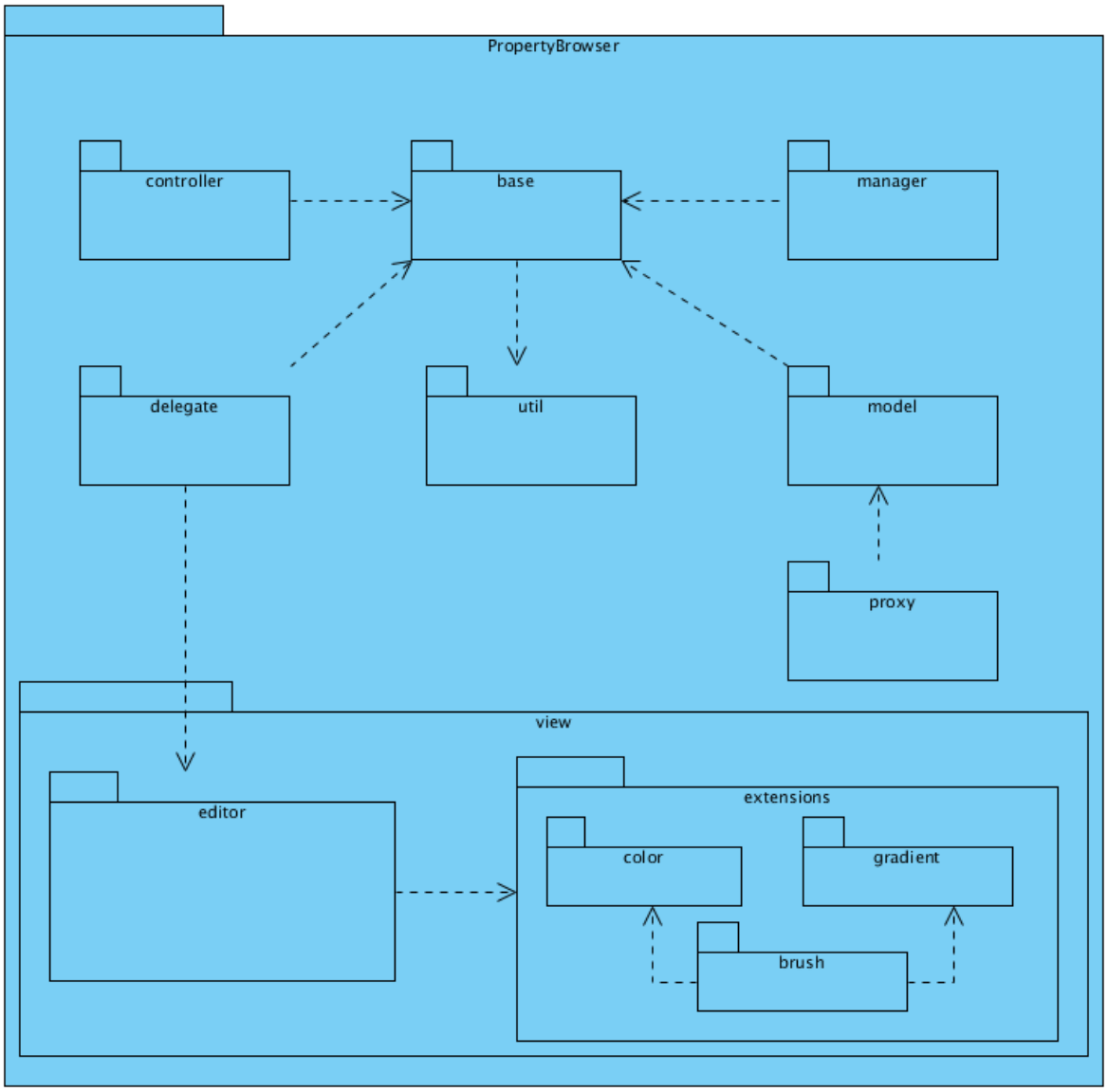


Figura 9. Diagrama de paquetes.

2.2.1 Descripción de los paquetes asociado a la solución propuesta

Paquete	Descripción
base	Corresponde a las clases abstractas que serán reimplementadas en la solución propuesta.
manager	Corresponde a la gestión de los diferentes tipos de datos utilizados en la solución propuesta.
util	Corresponde a la manipulación, conversión de datos utilizados en la solución propuesta.

model	Corresponde a la representación de las propiedades pertenecientes al objeto a inspeccionar por la solución propuesta.
controller	Corresponde a la creación de las propiedades así como el gestor que manipula dicho tipo de dato en la solución propuesta.
delegate	Corresponde al renderización a partir de los *widgets*[1] registrados en dependencia del tipo de dato a visualizar en la solución propuesta.
Proxy	Corresponde a la representación del modelo utilizado en la solución propuesta.
view	Corresponde a la visualización de los datos en la solución propuesta.
editor	Corresponde a la visualización de cada tipo de dato registrado en la solución propuesta.
extensions	Corresponde a la extensión de algunos editores para una mejor interacción del usuario en la solución propuesta.
color	Corresponde a la visualización del tipo de dato QColor en la solución propuesta.
gradient	Corresponde a la visualización del tipo de dato QGradient en la solución propuesta.
brush	Corresponde correspondientes a la visualización del tipo de dato QBrush en la solución propuesta.

Tabla 3. Descripción de los paquetes asociado a la solución propuesta.

2.3 Requisitos del sistema

Un requisito es una "*condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado*". También se aplica a las condiciones que debe cumplir o poseer un sistema o uno de sus componentes para satisfacer un contrato, una norma o una especificación. Existen dos tipos de requisitos, los funcionales y los no funcionales (Laguna, 2012).

A continuación se listan los requisitos que el sistema debe cumplir, además los requisitos funcionales estarán expresados y presentados mediante las Historias de

[1] Widgets: formularios, botones, menús, cuadros de texto

usuario (HU). Las HU son descripciones cortas redactadas en el lenguaje del usuario, donde este determina su punto de vista respecto a las necesidades de la aplicación.

2.3.1 Requisitos funcionales

Según Somerville, los requerimientos funcionales de un sistema describen lo que el sistema debe hacer. Estos requerimientos dependen del tipo de *software* que se desarrolle, de los posibles usuarios del *software* y del enfoque general tomado por la organización al redactar requerimientos. Cuando se expresan como requerimientos de usuario, habitualmente se describen de una forma bastante abstracta. Sin embargo, los requisitos funcionales del sistema describen con detalle la función de éste, sus entradas y salidas, excepciones, etcétera (Sommerville p. 2007).

Sirven de base para estimar el coste, el tiempo necesario para desarrollar el sistema y para planificar los contenidos técnicos de las iteraciones posteriores. A partir de la descripción de los procesos de negocios, se identificaron los requisitos a cumplir por el sistema, los cuales se muestran a continuación.

No.	Requisitos
RF1	Crear controlador para la manipulación de las propiedades del objeto.
RF2	Crear delegado para el renderización de las propiedades del objeto.
RF3	Crear modelo para las propiedades del objeto.
RF4	Permitir la gestión del tipo de dato Bool.
RF5	Modificación del tipo de dato Bool.
RF6	Visualización del tipo de dato Bool.
RF7	Permitir la gestión del tipo de dato Enum.
RF8	Modificación del tipo de dato Enum.
RF9	Visualización del tipo de dato Enum.
RF10	Permitir la gestión del tipo de dato Group.
RF11	Visualización del tipo de dato Group.
RF12	Permitir la gestión del tipo de dato Int.
RF13	Modificación del tipo de dato Int.
RF14	Visualización del tipo de dato Int.
RF15	Permitir la gestión del tipo de dato QBrush.
RF16	Modificación del tipo de dato QBrush.
RF17	Visualización del tipo de dato QBrush.
RF18	Permitir la gestión del tipo de dato QPen.
RF19	Modificación del tipo de dato QPen.

RF20	Visualización del tipo de dato QPen.
RF21	Permitir la gestión del tipo de dato QDate.
RF22	Modificación del tipo de dato QDate.
RF23	Visualización del tipo de dato QDate.
RF24	Permitir la gestión del tipo de dato QTime.
RF25	Modificación del tipo de dato QTime.
RF26	Visualización del tipo de dato QTime.
RF27	Permitir la gestión del tipo de dato QDateTime.
RF28	Modificación del tipo de dato QDateTime.
RF29	Visualización del tipo de dato QDateTime.
RF30	Permitir la gestión del tipo de dato String.
RF31	Modificación del tipo de dato String.
RF32	Visualización del tipo de dato String.
RF33	Permitir la gestión del tipo de dato QFont.
RF34	Modificación del tipo de dato QFont.
RF35	Visualización del tipo de dato QFont.
RF36	Permitir la gestión del tipo de dato QColor.
RF37	Modificación del tipo de dato QColor.
RF38	Visualización del tipo de dato QColor.
RF39	Filtrar propiedades por nombre.

Tabla 4. Requisitos funcionales.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales, como su nombre sugiere, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como fiabilidad, el tiempo de respuestas y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de dato que se utilizan en las interfaces del sistema (Sommerville p. 2007).

Usabilidad:

RNFU 1: El módulo debe presentar un acceso fácil y rápido, para facilitar su uso por los usuarios.

- **Diseño e implementación:**

RNFDI 1: Se utilizará como IDE Qt Creator en su versión 3.3.1 para el desarrollo del sistema.

RNFDI 2: Se utilizará Visual Paradigm en su versión 8.0 como herramienta de modelado.

❖ **Funcionamiento:**

- **Software:**

RNFS 1: Para el correcto funcionamiento en la PC será necesario tener instalado el sistema operativo GNU Linux.

- **Hardware:**

RNF 1: La PC para la correcta ejecución de la aplicación deberá tener las siguientes características mínimas:

- Procesador Intel Pentium Dual Core a 2.0 Ghz, equivalente o superior.
- 2GB[2] de memoria RAM[3].
- Capacidad de 80 GB de HDD[4].

❖ **Fiabilidad:**

RNF 1: Ante cualquier operación que se realice sobre el módulo, este debe responder en no más de 6 segundos.

2.3.3 Historia de Usuario

Las HU son utilizadas como herramientas para dar a conocer los requerimientos del sistema al equipo de desarrollo. Son pequeños textos para describir una actividad que realizará el *software*. Se puede considerar que estas juegan un papel similar a los casos de uso en otras metodologías, pero en realidad son muy diferentes porque solo muestran la silueta de una tarea a realizarse.

Las HU también son utilizadas para estimar el tiempo que el equipo de desarrollo tomará para realizar las entregas. En una entrega se puede desarrollar una o más HU, esto depende solo del tiempo que demore la implementación de cada una (Pressman, 2007). Como ejemplo se pone la HU permitir la gestión del tipo de dato Bool, evidenciándose en la Tabla 5 HU1. (Ver **Anexo 1: Historias de Usuario)**

[2] GB: Gigabyte, es una unidad de almacenamiento.
[3] RAM: Memoria de Acceso Aleatorio- Random Access Memory.
[4] HDD: Unidad de Disco Duro- Hard Disk Drive.

Historia de Usuario	
Número: 1.	**Nombre**: Permitir la gestión del tipo de dato Bool.
Programador: Yoan Díaz Zayas	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Media
Puntos Estimados: 0.20	**Iteración asignada:**1
Puntos Reales: 0.20	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato Bool.	
Observaciones:	

Tabla 5. HU1.

2.4 Patrón Arquitectónico

La selección de un patrón arquitectónico es una decisión fundamental de diseño en el desarrollo de un sistema de *software* debido a que provee un conjunto de subsistemas predefinidos. Además especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos (Camacho, 2004).

En el desarrollo de un *software*, se hace necesario seleccionar diferentes patrones los cuales ayudan a que esté presente una buena estructura y organización que hace eficiente su funcionamiento. Estos patrones son una guía para cometer una determinada acción. Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones, para organizar los distintos componentes.

La solución propuesta sigue el patrón arquitectónico Modelo Vista Controlador (MVC) el cual hace una separación de la interfaz de usuario de la lógica de control de una aplicación. Por su parte las vistas se encargan de la interacción y presentación de la información al usuario, mientras que los modelos se encargan de controlar y proporcionar a las vistas los datos necesarios para la correcta manipulación de los valores de las propiedades del objeto inspeccionado.

2.4.1 Modelo Vista Controlador y biblioteca gráfica del framework Qt

Framework Qt introdujo un conjunto de clases que posibilitan implementar la arquitectura MVC para gestionar las relaciones entre los datos y su forma de presentación al usuario. La separación de las funcionalidades introducidas por esta arquitectura ofrece a los desarrolladores una mayor flexibilidad para personalizar la presentación de los datos y una interfaz estándar de acceso a los datos.

2.4.2 Elementos de la arquitectura Modelo vista Delegado en *framework* Qt

- **Modelo:** Es el encargado de comunicarse con la fuente de datos, proporciona una interfaz para los otros componentes en la arquitectura. La naturaleza de la comunicación depende del tipo de fuente de datos y la forma en que se implementa el modelo.
- **Vista:** Obtiene los índices del modelo, estos constituyen referencias a los elementos de datos. Mediante los índices del modelo la vista puede recuperar los elementos de datos de la fuente de datos.
- **Delegado:** En una vista estándar un delegado representa los elementos de datos, cuando un elemento es editado el delegado es el encargado de comunicarse directamente con el modelo usando los índices. En la Figura 10 se muestra el modelo vista delegado.

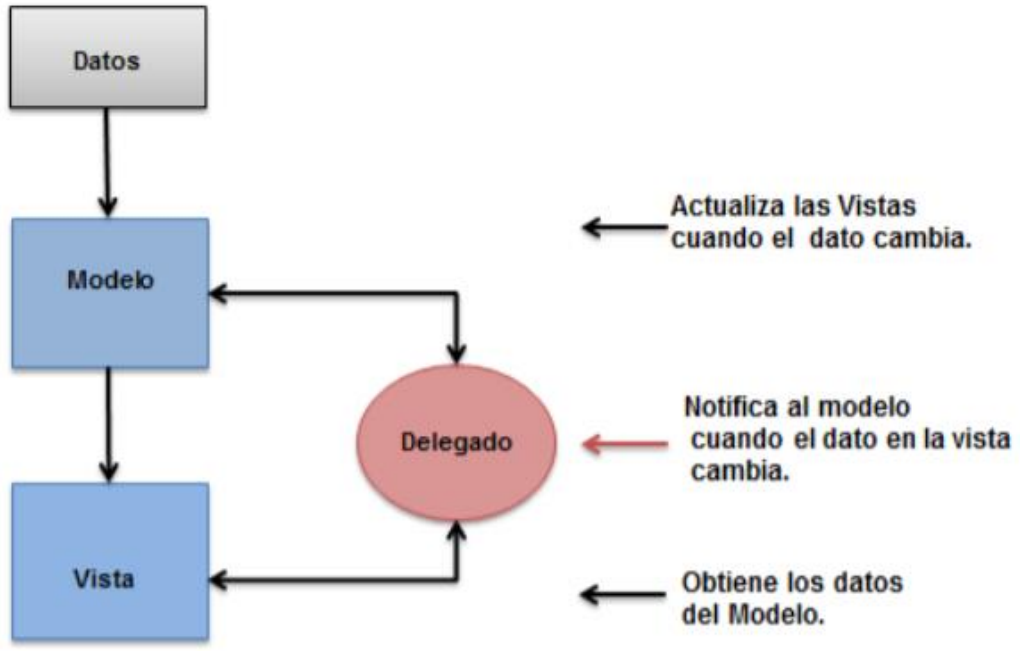


Figura 10. Modelo Vista Delegado (MVD).

Los modelos, vistas y delegados se comunican entre sí mediante *signals (señales) y slosts*:

- Las *signals* del modelo informan a la vista sobre los cambios en los datos mantenidos por la fuente de datos.
- Las señales procedentes de la vista deben proporcionar información acerca de la interacción del usuario con los elementos que se muestran.
- Las señales procedentes del delegado se utilizan durante la edición para indicar al modelo y a la vista sobre el estado del editor o contenedor del dato.

2.5 .Diagrama de Clases

El diagrama de clases expresa la estructura u organización del *software* en términos de las clases. Es un reflejo abstracto de los componentes y las relaciones entre ellos. Estos diagramas son el pilar básico del modelado con UML, utilizado tanto para mostrar lo que el sistema puede hacer, como para mostrar cómo puede ser construido. Cuando se crean, se modela una parte de los elementos y de las relaciones que configuran la vista del diseño del sistema. A continuación se muestra el diagrama de clases del diseño que representa la estructura del sistema, las entidades, sus atributos y relaciones. En la Figura 11 se muestra el diagrama de clases.

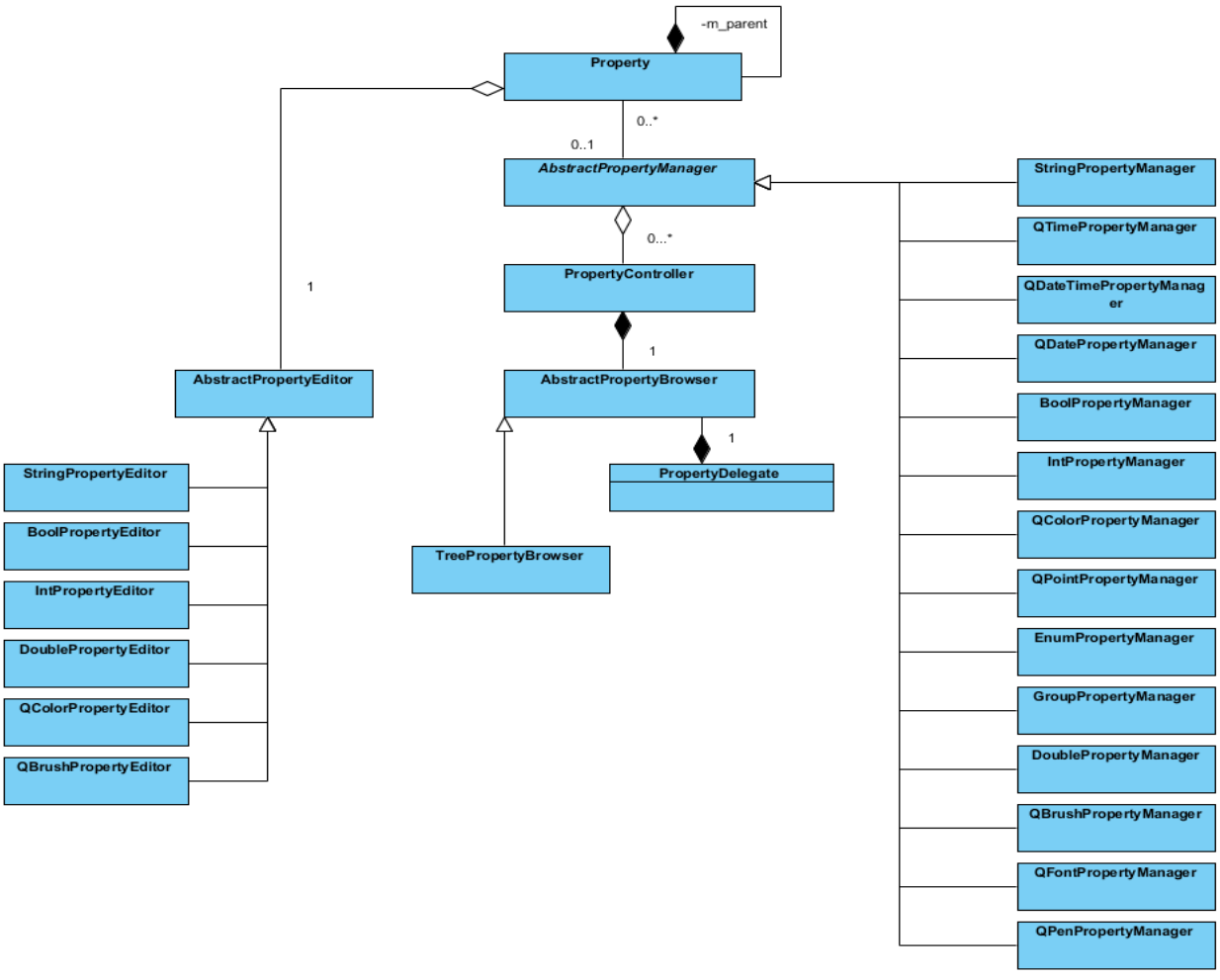


Figura 11. Diagrama de Clases.

2.6 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces. Son soluciones a diferentes clases de problemas conocidos, que pueden ser aplicadas a diferentes códigos (Rivero, 2013). El uso de estos patrones permite ahorrar grandes cantidades de tiempo en la construcción de *software*, tener una

estructura de código común para todos los proyectos que implementen una funcionalidad genérica y es más fácil de comprender, mantener y extender un *software* que aplique patrones de diseño (Gamma, 1995).

2.6.1 Patrones GoF[5]

Los patrones GoF se descubren como una forma indispensable de enfrentarse a la programación. Cuando se publica el libro "Design Patterns: Elements of Reusable Object Oriented Software" escrito por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, la idea de patrones de diseño comenzó a tomar fuerza. Ellos recopilaron y documentaron 23 patrones de diseño, los que proporcionan elementos reusables en el diseño de sistemas de *software*, por tanto se pueden aplicar a diferentes problemas de diseño en distintas circunstancias (García, 2012). (Ver **Anexo 3: Patrones GoF**). Estos se utilizan durante la implementación del sistema con el objetivo de poder solucionar los problemas que suelen ser comunes en el desarrollo de *software*. Estos se pueden clasificar en:

- **Creacionales:** Los patrones creacionales abstraen el proceso de creación de instancias y ocultan los detalles de cómo los objetos son creados o inicializados.
- **Estructurales:** Los patrones estructurales se ocupan de cómo las clases y objetos se combinan para formar grandes estructuras y proporcionar nuevas funcionalidades.
- **Comportamiento:** Los patrones de comportamiento están relacionados con los algoritmos y la asignación de responsabilidades entre los objetos. Son utilizados para organizar, manejar y combinar comportamientos.

A continuación se describen los que se seleccionaron para el desarrollo de la solución propuesta:

- **Plantilla:** Define en una operación, el esqueleto de un algoritmo, delega en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.
- **Proxy:** Define un representante o sustituto de otro objeto para controlar el acceso a éste, con el objetivo de retrasar el coste de creación e inicialización de un objeto hasta que sea realmente necesario.

[5] GoF: Gang of Four, Pandilla de los Cuatro.

El patrón plantilla es usado en las entidades que heredan de AbstractPropertyManager que redefinen ciertos pasos en los algoritmos de manipulación de los datos de las propiedades, en la figura 12 se muestra el uso de dicho patrón.

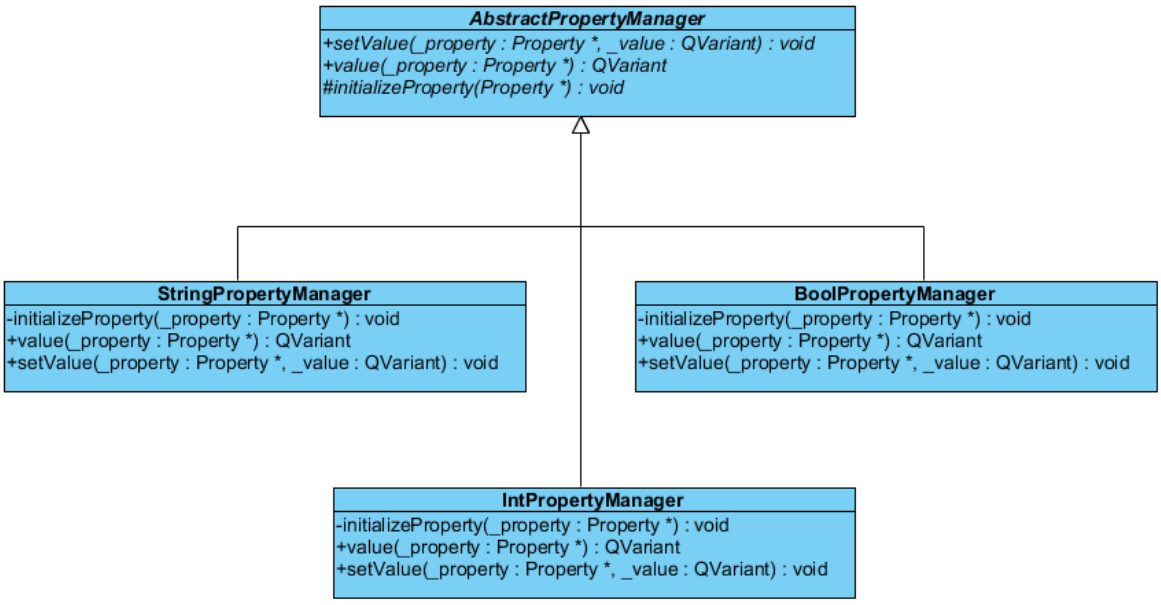


Figura 12. Patrón plantilla.

El patrón *proxy* es usado para el filtrado de las propiedades por el nombre a partir de la entidad TreePropertyProxyModel, en la figura 13 se muestra el uso de dicho patrón.

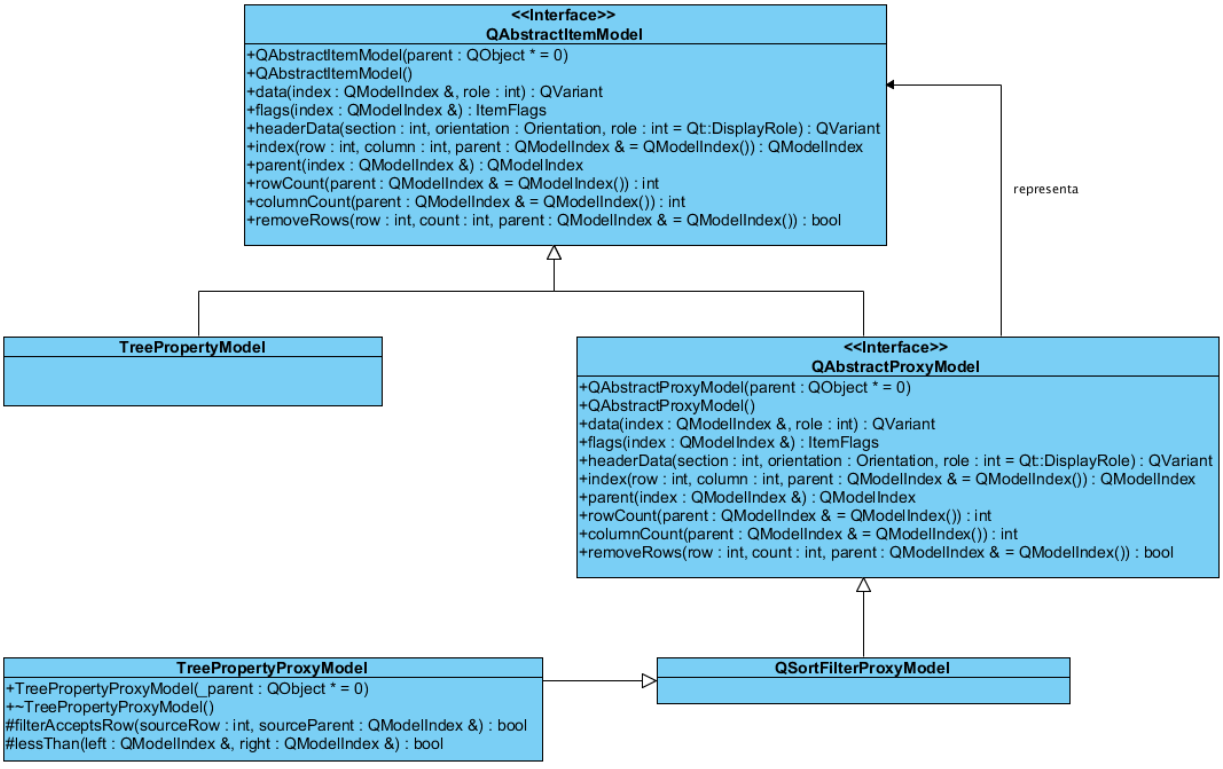


Figura 13. Patrón proxy.

2.6.2 Patrones Generales de Software para Asignar Responsabilidades (GRASP[6])

Estos describen los principios fundamentales de la asignación de responsabilidades a las clases y objetos. Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de *software* (Cortés, 2010). Para el diseño de la aplicación de monitoreo se tienen en cuenta los patrones GRASP, los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. A continuación se muestra una selección de estos patrones los cuales serán utilizados durante la realización del diseño de la aplicación:

- **Experto**

Es el patrón encargado de asignar responsabilidades; expresa que siempre se debe asignar una responsabilidad al experto en información, o sea, a la clase que cuenta con la información necesaria para llevar a cabo la funcionalidad. Este patrón se pone de manifiesto en la clase Property, la cual contiene los atributos de cada propiedad del objeto a inspeccionar:

Figura 14. Patrón experto.

[6] GRASP: General Responsibility Assignment Software Patterns

❖ **Creador**

Permite asignar el responsable de la creación de una nueva instancia de alguna clase. Explica qué clase es la encargada de crear objetos, en determinados escenarios de ejecución y guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito general de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Este patrón se evidencia en la clase AbstractPropertyManager, específicamente en el método addProperty, encargado de crear una propiedad.

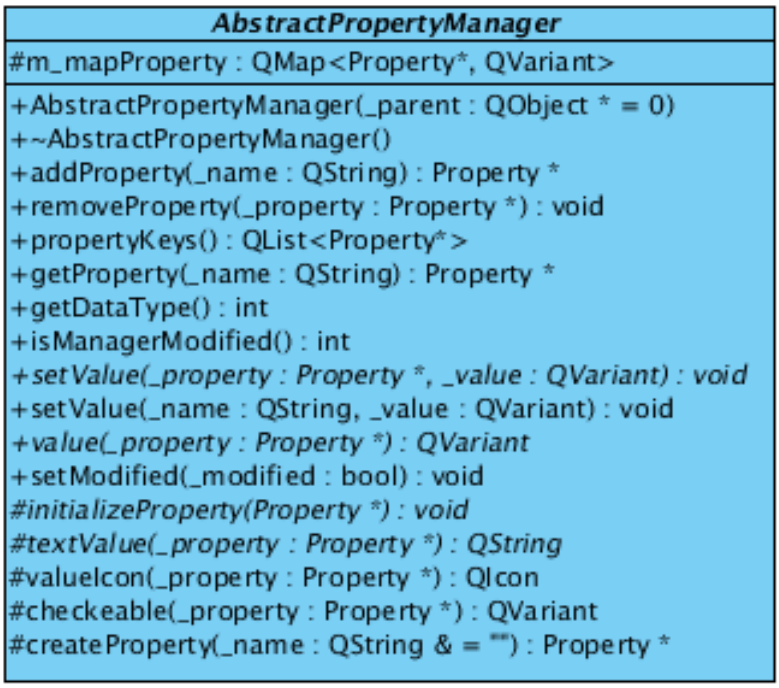


Figura 15. Patrón creador.

❖ **Alta cohesión**

Sigue el principio de que cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable. A través de este patrón se simplifica el mantenimiento y las mejoras del funcionamiento del sistema. Las clases que están sobrecargadas de métodos poseen una alta cohesión por lo que se recomienda para un buen diseño la creación de los paquetes de servicio o clases agrupadas por funcionalidades que son fácilmente reutilizables.

Bajo acoplamiento

Permite impulsar la asignación de responsabilidades de manera que su localización no incremente el acoplamiento hasta un nivel que lleve a los resultados negativos

que puede producir un acoplamiento alto. No soporta el diseño de clases que son más independientes, lo que reduce el impacto del cambio. Él no se puede considerar de manera aislada a otros patrones como el Experto o el de Alta cohesión, sino que necesita incluirse como uno de los diferentes principios de diseño que influyen en una elección, al asignar una responsabilidad.

2.7 Conclusiones parciales

En este capítulo se abordaron los aspectos fundamentales del análisis y diseño de la propuesta de solución. El levantamiento de los requerimientos del sistema permitió determinar las funcionalidades básicas a desarrollar durante el proceso. A raíz de esto se determinaron 39 requisitos funcionales y 6 requisitos no funcionales. Además de los patrones de diseño, de los cuales se utilizaron los Graps y los GoF; además se utilizó el patrón arquitectónico MVD.

CAPÍTULO 3: Implementación y pruebas

Introducción

El presente capítulo describe los componentes internos y externos de integración del sistema a través del diagrama de componentes y detalla el diagrama de despliegue cuya integración define el modelo de implementación. Se explica el proceso de realización y aplicación de las pruebas al sistema, se analizan las no conformidades encontradas en cada una de las iteraciones realizadas. Se valora el producto de acuerdo a la seguridad, usabilidad, portabilidad, fiabilidad y confiabilidad.

3.1 Diagrama de componentes

Un componente es una parte física de un sistema que materializa una o más clases al ser una abstracción con atributos y métodos que pueden ser implementados en ellos. En la figura16 se muestra el diagrama de componentes del sistema, haciendo énfasis los servicios externos e internos que consume y los servicios que brinda.

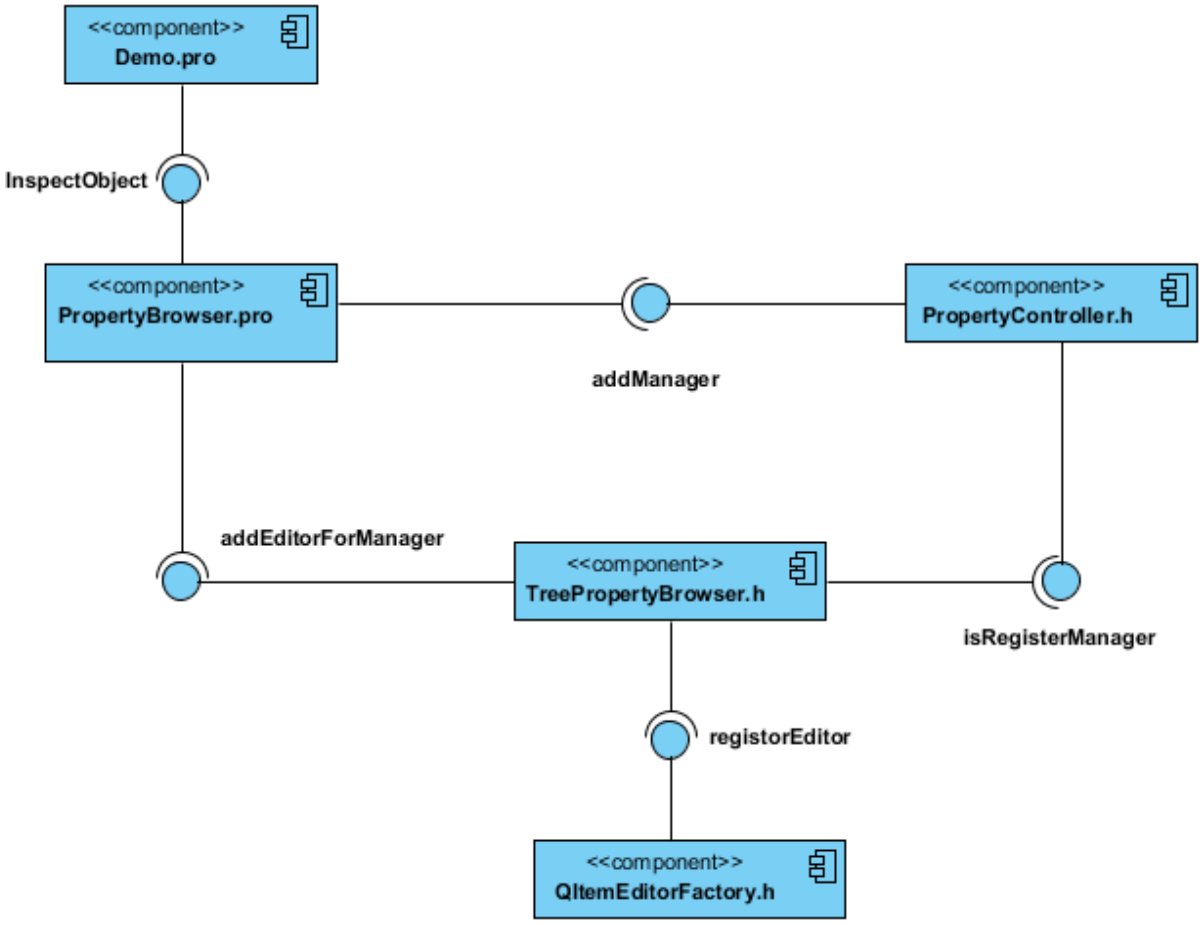


Figura 16. Diagrama de componentes.

3.2 Integración de la solución

La solución propuesta está destinada ser una biblioteca, con el objetivo de encapsular clases y rutinas útiles para ser reutilizadas en el código fuente de un programa, lo cual conlleva a la escritura de instrucciones una sola vez y luego hacer referencias a ellas desde aplicaciones que necesiten esas funcionalidades. A continuación se hace necesario evidenciar la correcta utilización del inspector de propiedades desarrollado. Para mostrar la interfaz visual del componente se necesita hacer referencia a la clase *PropertyBrowserManager* mediante la interfaz *initialize*, encargada de crear el controlador de las propiedades y el registro de los manejadores de los tipos de datos que no hereden de la clase QObject definidos por el *framework* Qt, para la edición de las propiedades se usa la interfaz *registerEditor* donde se especifica el tipo de dato y la interfaz visual que representará el valor de la propiedad seleccionada para realizar la inspección de las propiedades de un objeto se utiliza la interfaz *inspectObject*, en la figura 17 se muestra la representación del tipo de dato *QColor* en el inspector de propiedades.

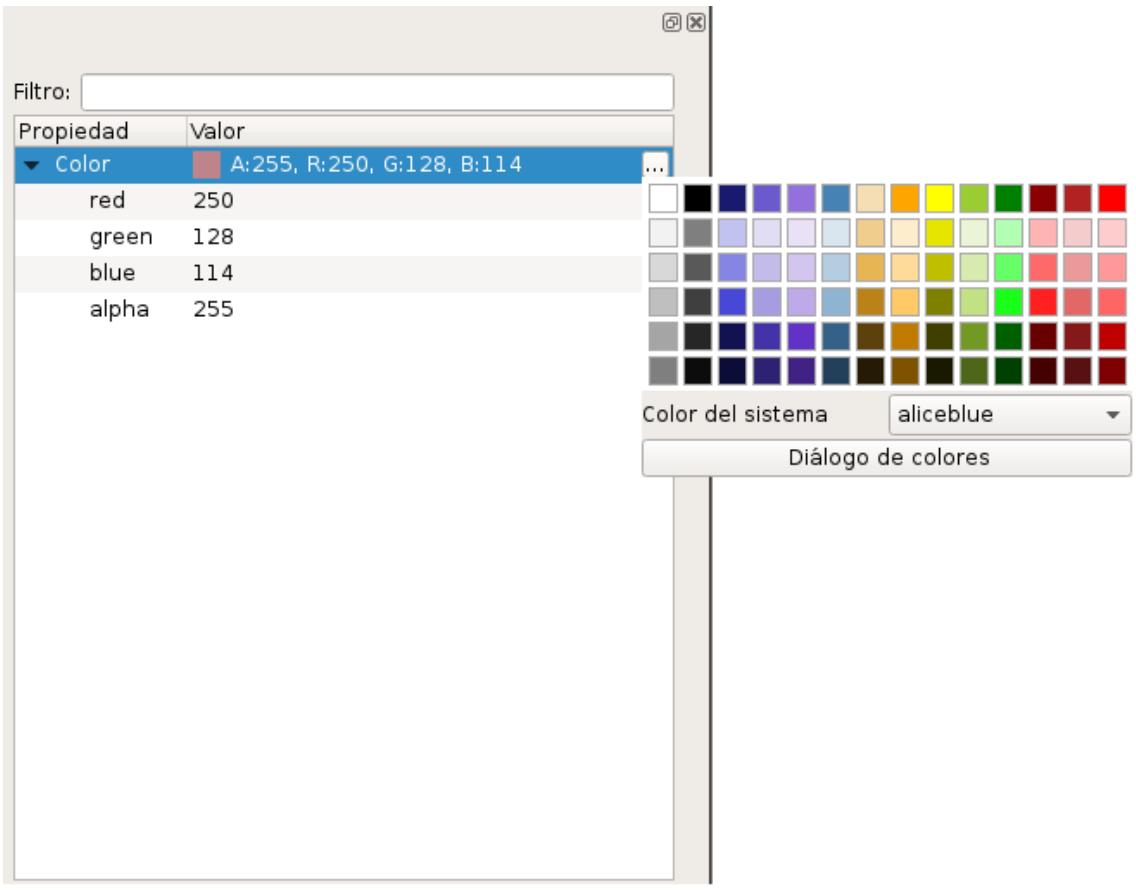


Figura 17 Representación del tipo de dato QColor en el inspector de propiedades.

3.3 Resultados del trabajo

Como resultado del trabajo de obtuvo un componente que se puede integrar a las soluciones del CEDIN con tecnología Qt en su versión 5. Con la creación de este componente se mejoró el modo de visualización de las funcionalidades que presentaba el anterior inspector .En la figura 18 se muestra la representación del tipo de dato *QBrush* en el inspector de propiedades. Para observar el producto logrado (Ver **Anexo 4: Inspector de propiedades)**

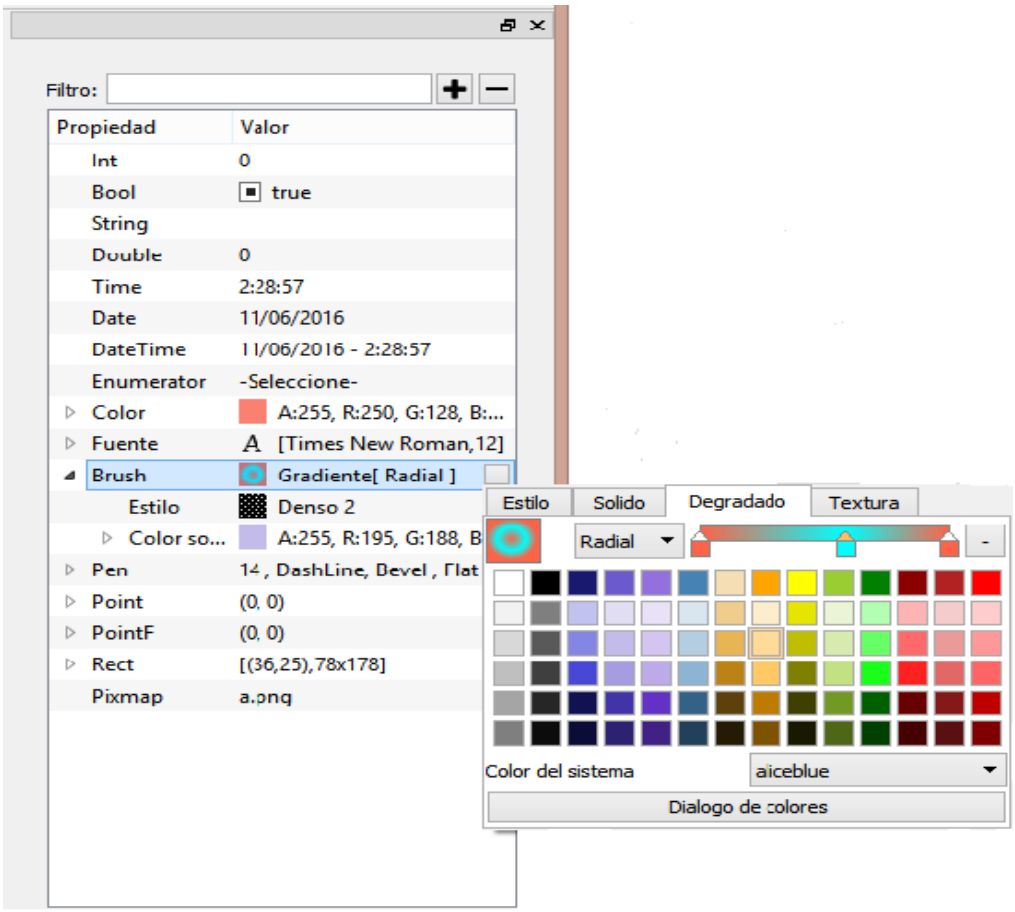


Figura 18. Representación del tipo de dato QBrush en el inspector de propiedades.

3.4 Estándar de codificación

El estándar de codificación utilizado para el desarrollo de este producto de *software*, permite establecer una serie de reglas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código.

A continuación se describen algunos de los elementos fundamentales que componen el estándar utilizado:

Nombres:

- Los nombres de las clases son sustantivos singulares.
- Los nombres deben reflejar el qué y no el cómo.
- Los nombres no deben revelar detalles de implementación.
- Escoger nombres lo suficientemente largos para ser expresivos, pero evitar manejar nombres que dificulten la labor de implementación.
- Evitar redundancia no repetir nombres de clases en sus elementos.
- Los nombres de los atributos y parámetros son frases con sustantivos.

Manejo de errores:

- Se pueden manejar errores mediante mecanismos de excepciones o mediante valores de retorno, aunque esto debe ser uniforme dentro de un mismo objeto.
- Es buena práctica emplear herramientas para identificar errores en la codificación en tiempo de ejecución.

Documentación y Comentarios:

- Documentar mientras se programa.
- Documentar eliminación de errores y cambios sobre el código
- Al modificar el código se deben actualizar todos los comentarios y documentación asociada.
- Evitar agregar comentarios al final de líneas de código, salvo en el caso de declaraciones. En este caso, tales comentarios deben estar alineados.
- Antes de la entrega de la aplicación, eliminar todos los comentarios superfluos[7] y/o temporales con la finalidad de evitar confusiones en su mantenimiento.

Codificación:

- Inicializar todas las variables.
- Emplear líneas en blanco para separar pasos lógicos (declaraciones, lazos).
- Comentar siempre las llaves que cierran.
- Emplear constantes en sustitución de números o cadenas de caracteres literales.
- Minimizar el alcance de las variables para evitar confusión y facilitar el mantenimiento.
- Emplear líneas en blanco para organizar el código, para crear párrafos de código para una mejor lectura.

3.5 Fase de prueba

Se escogen algunas de las pruebas planteadas por (Pressman p. 2007) para aplicarlas al módulo implementado. Este autor propone como pauta a seguir que el proceso debe

[7] Superfluos: innecesario, sobrante.

realizarse desde la parte más pequeña del *software* hasta la más grande, para formar una espiral entre los niveles de prueba. El objetivo fundamental de las pruebas es determinar si el código generado funciona correctamente y realiza las operaciones deseadas. Además permiten comprobar que el producto cumple con los requisitos acordados con el cliente. Por último, verifican que el sistema sea capaz de funcionar normalmente en un ambiente parecido o en el ambiente de despliegue.

Para cumplir con lo antes mencionado se escaló desde las pruebas unitarias (encargadas de probar el código), hasta las pruebas de aceptación (encargadas de probar los requisitos).

3.5.1 Pruebas unitarias o Prueba de caja blanca

Las pruebas unitarias se centran en un esfuerzo de verificación de la unidad más pequeña del diseño de *software*: el componente y el módulo del *software*. *"Las pruebas de unidad se centran en la lógica del procesamiento interno y en las estructuras de datos dentro de los límites de un componente."* (Pressman p. 2007).

Las pruebas de la caja blanca se realizan sobre las funciones de un módulo en concreto, están dirigidas a las funcionalidades internas. Entre las técnicas usadas se encuentran; prueba del camino básico, pruebas de condición y pruebas de bucles.

Para la realización de las pruebas se utilizó el método de caja blanca se seleccionó la técnica del camino básico, la cual permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. En la figura19 se muestra la representación de las pruebas de caja blanca.

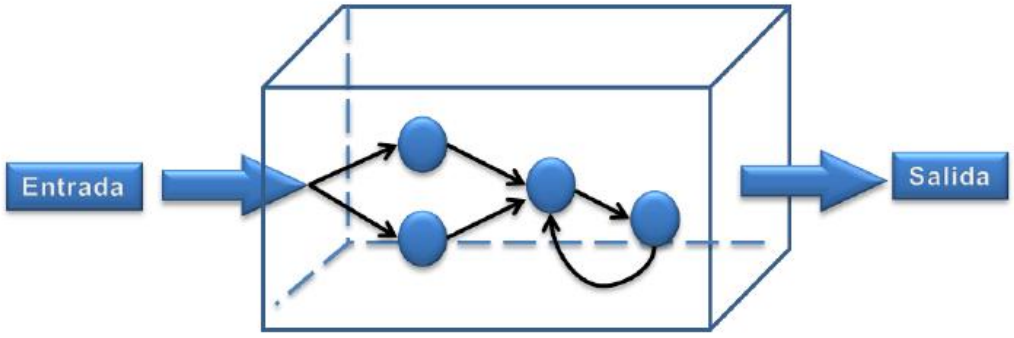


Figura 19. Representación de las pruebas de caja blanca.

Para aplicar la técnica del camino básico se debe introducir la notación para la representación del flujo de control, este puede representarse por un Grafo de Flujo en el cual:

- Cada nodo del grafo corresponde a una o más sentencias de código fuente.
- Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo.
- Se calcula la complejidad ciclomática del grafo.

Para construir el grafo se debe tener en cuenta la notación para cada una de las instrucciones.

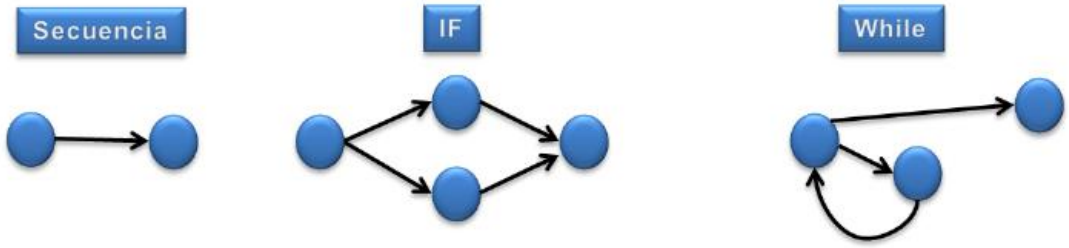


Figura 20. Notación de grafos de flujo.

Un grafo de flujo está formado por 3 componentes fundamentales que ayudan a su elaboración y comprensión, estos brindan información para confirmar que el trabajo se está haciendo adecuadamente.

Componentes del grafo de flujo:

Nodo: Son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.

Aristas: Son constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aun cuando el nodo no representa la sentencia de un procedimiento.

Regiones: son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran, la cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un procedimiento.

A continuación, se analizan y enumeran las sentencias de código del método **textValue** contenidos en la clase **QColorPropertyManager**. Este método permite visualización el valor que toma cada uno de los colores según sean modificados por el usuario. En la *Figura 21* se detalla el código del método textValue.

```
QString QColorPropertyManager::textValue(Property *_property)
{
    QString t_textColor = "";                                          1
    if(m_mapProperty.contains(_property))                              2
    {
        QColor t_color = value(_property).value<QColor>();
        t_textColor = QString("A:%1, R:%2, G:%3, B:%4 ")               3
                .arg(t_color.alpha())
                .arg(t_color.red())
                .arg(t_color.green())
                .arg(t_color.blue());

    }                                                                  4
    return t_textColor;
}
```

Figura 21. Método textValue.

Luego, es necesario representar el grafo de flujo asociado al código antes presentado a través de nodos, aristas y regiones, ver figura 22.

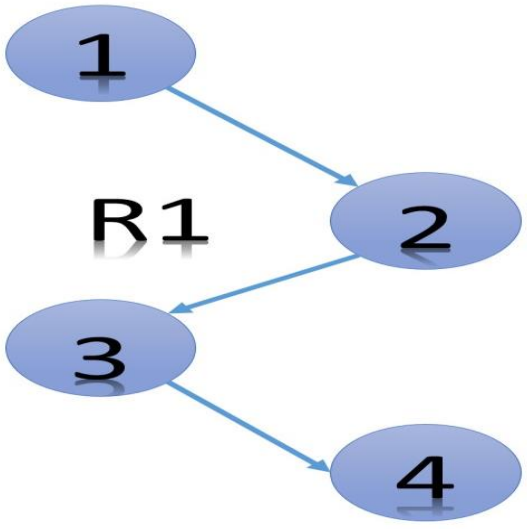


Figura 22. Grafo de flujo asociado al método textValue.

Una vez construido el grafo de flujo asociado al procedimiento anterior, se determina la complejidad ciclomática, la cual es una métrica de *software* muy útil, pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar.

Cálculo de la complejidad ciclomática para el grafo de flujo de la Figura 22

V (g) = (a-n) + 2 **(I)** V (g) = (p+1) **(II)** V (g) = r **(III)**

El "*V (g)"* es el valor de la complejidad ciclomática "a" la cantidad total de aristas, "n" la cantidad total de nodos, "p" la cantidad total de nodos predicados (nodos de los cuales parten dos o más aristas) y "r" la cantidad total de regiones, se incluye el área exterior del grafo como una región más.

V (g) = (3-4) + 2 = 1 **(I)** V (g) = 0 + 1 = 1 **(II)** V (g) = 1 **(III)**

La evaluación de las fórmulas I, II y III arroja un valor de complejidad ciclomática igual a 1, de manera que existen 1 posibles caminos por donde el flujo puede circular. Este valor representa el número mínimo de casos de pruebas para el procedimiento tratado. Seguidamente, es necesario especificar los caminos básicos que puede tomar el algoritmo durante su ejecución.

- Camino básico #1: 1 – 2 – 3 – 4

Se procede a ejecutar los casos de pruebas para cada uno de los caminos básicos determinados en el grafo de flujo, ver *Tablas 15 y 16.* Para definir los casos de prueba es necesario tener en cuenta:

- **Descripción:** Se describe el caso de prueba y se especifican los aspectos fundamentales de los datos de entrada.
- **Condición de ejecución:** Se verifica que cada parámetro cumpla las condiciones de ejecución.
- **Entrada:** Se muestran los parámetros de entrada del procedimiento.
- **Resultados esperados:** Se especifica el resultado que debe arrojar el procedimiento.

Caso de Prueba Camino Básico #1
Descripción: La Variable "color" contiene los datos definidos para el formulario de color en la aplicación. ❖ Rojo: Valor numérico que desea (0… 255).

❖ Azul: Valor numérico que desea (0… 255). ❖ Verde: Valor numérico que desea (0… 255). ❖ Alpha: Valor numérico que desea (0… 255).
Condiciones: Ejecutar aplicación.
Entrada: ❖ Rojo: 160 ❖ Azul: 150 ❖ Verde:10 ❖ Alpha: 0
Resultado Esperado: El sistema debe de actualizar los valores mediante el usuario los modifica.
Resultado: Satisfactorio.

Tabla 6. Caso de prueba camino básico#1.

Resultados

Se probaron, mediante casos de pruebas de caja blanca o Pruebas Unitarias (PU) como también se le conocen, algunas de las funcionalidades más críticas del sistema como fue en el caso de **ColorPropertyManger**. Los casos de PU aplicados obtuvieron resultados satisfactorios, que representa el 100% de los realizados.

3.5.2 Prueba de Aceptación.

"La validación del *software* se logra mediante una serie de pruebas que demuestren que se cumple con los requisitos al construir un *software* personalizado para un cliente se aplica una serie de pruebas de aceptación que permiten al cliente validar todos los requisitos" (Pressman p. 2007). Debido que la solución propuesta será utilizada por un número creciente de usuarios con distintos roles dentro del negocio (técnicos, especialistas, proveedores y clientes del centro de soporte) no es práctico realizar pruebas de aceptación formales para cada uno. "La mayoría de los constructores de productos de *software* emplean procesos llamados prueba alfa y prueba beta para descubrir errores que solo el usuario final podría detectar" (Pressman p. 2007). Las pruebas alfa se realizan en el lugar de trabajo del desarrollador, de esta manera se crea un ambiente controlado donde los usuarios finales pueden trabajar con el sistema y registrar los errores y problemas.

Pruebas de aceptación al sistema.

A continuación, se muestran algunos casos de pruebas relacionados con las historias de usuarios. Las restantes podrán ser consultadas en los anexos. (Ver **Anexo 2: Prueba de Aceptación.).**En la tabla 7 se evidencia la prueba de aceptación#1.

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de usuario:1**
Nombre: Permitir la gestión del tipo de dato Bool.	
Descripción: El sistema debe visualizar el tipo de dato Bool, permitir que sea modificado.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato Bool.	

Tabla 7. Prueba de aceptación#1.

Resultados

De un total de 14 casos de PA, 3 de ellos resultaron no satisfactorios, lo cual representa el 21% del total de casos de pruebas realizadas. Mientras que los 10 restantes resultaron satisfactorios para un 79%. Los errores detectados por los casos de pruebas no satisfactorios fueron mitigados después de 2 iteraciones de prueba.

3.6 Conclusiones parciales.

En este capítulo se mostró el diagrama de componentes, que con él se tiene una mejor visión para su posterior implementación. Se estableció el estándar de codificación a utilizar lo que permitió establecer pautas para mejorar el desarrollo del código. Por último se establecieron las pruebas de caja blanca que se realizaron sobre las funciones de un módulo en concreto, están dirigidas a las funcionalidades internas, además en las pruebas de aceptación se comprobó que la implementación realizada se ajustaba a lo especificado en las HU. De esta manera se concluyó el proceso de desarrollo y se determinó que el producto generado cumple con todas las exigencias.

CONCLUSIONES GENERALES

Para el trabajo con aplicaciones donde se maneja un gran número de entidades, la edición de propiedades se vuelve un requisito con mucho peso que deben poseer los sistemas de este tipo, constituye una técnica de vital importancia para presentar las características de cada entidad al usuario y que este pueda editarlas. Durante el desarrollo de la investigación se cumplieron los objetivos propuestos, concluyéndose que:

- Se desarrolló un componente para la gestión de propiedades de las entidades que se puede integrar a las soluciones del CEDIN con tecnología Qt.
- Mediante varias iteraciones de pruebas al *software* y luego de solventar las no conformidades encontradas se pudo satisfacer correctamente los requisitos funcionales planteados para el inspector.
- Con el desarrollo del componente inspector de propiedades se contribuye a la soberanía tecnológica.

RECOMENDACIONES

Referente a los resultados obtenidos en la investigación efectuada durante la elaboración de este trabajo, y con la intención de asegurar la posterior ampliación, modificación y mejora del inspector de propiedades propuesto, se exponen a continuación algunas recomendaciones:

- Agregar al producto la posibilidad de permitir la gestión de propiedades dinámicas.
- Desarrollar o implementar nuevas entidades de tipos vistas que permita visualizar los modelos de forma (tabulada, Group).

BIBLIOGRAFÍA

1. Aguilar, Alfonso. 2010. Aprenda QT desde hoy mismo. *Aprenda QT desde hoy mismo.* [En línea] octubre de 2010. http://es.scribd.com/doc/120249264/Aprenda-Qt4-hoy-mismo#force_seo..
2. Albán, O.A.V. 2010. *Introducción a Qt y Qt Creator.* Cauca, Colombia : s.n., 2010.
3. Alegsa, Leandro. 2010. DICCIONARIO DE INFORMÁTICA Y TECNOLOGÍA. [En línea] 2010. http://www.alegsa.com.ar/Dic/ide.php.
4. Aris Goicoechea Lassaletta, Marcos Calleja Fernández, Pablo Pizarro Moleón. 2012. *Terraform: simulación de vida sobre.* 2012.
5. Autores. 2013. Lenguaje de programación C++. *Lenguaje de programación C++.* [En línea] 26 de marzo de 2013. [Citado el: 20 de enero de 2016.] http://lenguajedeprogramacion21.blogspot.com/..
6. Autores, C.D. 2015. Qt Project. *Qt Project.* [En línea] 2015. [Citado el: 15 de enero de 2016.] <qt-project.org>.
7. Ayala. 2015. Utilización de los paneles de edición de Flash. *Utilización de los paneles de edición de Flash.* [En línea] 2015. [Citado el: 26 de enero de 2016.] https://helpx.adobe.com/es/animate/using/authoring-panels.html.
8. Camacho, Erika , Fabio Cardeso, Gabriel Nuñez. 2004. *Arquitecturas de software.* 2004.
9. Castaño, Ángela María R. 2007. Aprende en Línea. *Plataforma Académica para pregrado y posgrado.* [En línea] Univeridad de Antoquia, 16 de mayo de 2007. http://aprendeenlinea.udea.edu.co/lms/moodle/mod/resource/view.php?id=13232.
10. Cortés, Gloria. 2010. *INTRODUCCION A LOS PATRONES DE DISEÑO.* 2010.
11. 2008-2015. Definicion.de. [En línea] 2008-2015. http://definicion.de/lenguaje-de-programacion/.
12. Developer Network. 2015. Developer Network. *Developer Network.* [En línea] 2015. https://msdn.microsoft.com/es-es/library/cc437060%28v=vs.71%29.aspx.
13. DIGIA OYJ, V. 2013. Welcome to Qt. *Welcome to Qt.* [En línea] 2013. [Citado el: 15 de enero de 2015.] <http://qt.digia.com/>.

14. Flores, Ervin. 2012. *PROCESO UNIFICADO ÁGIL (AUP).* 2012.

15. Gamma, E. 1995. *Desing Patterns.* s.l. : Adison Wesley, 1995.

16. García, Luis Enrique. *Sistema SCADA.*

17. García, Sergio. 2012. Patrones de diseño GoF. [En línea] 2012. http://www.godtic.com/blog/2012/11/15/patrones-de-diseno-gof/.

18. Gutiérrez, Damián. 2009. *Diagrama de Paquete.* 2009.

19. —. 2009. *UML Diagramas de Paquetes (UML ilustrado).* 2009.

20. Hernán, Marcelo Ruiz. 2003. *Programación C.* Ciudad de Buenos Aires, Argentina : MP Ediciones S.A, 2003.

21. Hérnandez Muñoz, José Luis. 2006. Metodología de desarrollo de software. [En línea] 2006. http://www.um.es/docencia/barzana/IAGP/Iagp2.html.

22. Huanay Martínez, Franz. 2010. Tutorial Básico - Unity. *Tutorial Básico - Unity.* 2010.

23. JACOBSON, RUMBAUGH BOOCH. 2000. El Lenguaje Unificado de Modelado. Manual de Referencia. [aut. libro] 2000. *El Lenguaje Unificado de Modelado, Manual de Referencia.* Madrid : Pearson Educación.S.A.., 2000.

24. JOSÉ, ESTHER SALICHS SAN. 2012. *Desarrollo de un sistema HMI para un almacén automatizado.* Madrid : s.n., 2012.

25. Laguna, Miguel A. 2012. *Requisitos.* 2012.

26. Lalcebo, Yanelys del Rosario . 2013. *Subsistema de comunicaciones para el SCADA SAINUX.* 2013.

27. Pressman. *7ma Edición.* 2007 : s.n.

28. Pressman, Roger S. 2007. *Pressman.* 2007.

29. 2011. Qt Creator, IDE Overview. *Qt Creator, IDE Overview.* [En línea] 2011. https://qtproject.org/doc/qtcreator-2.5/creator-overview.html.

30. Rivero, Miguel A. Albuerne. 2013. *Simulador de dispositivos PLC-5 para realizar el proceso de.* La Habana : s.n., 2013.

31. Sánchez, Tamara Rodriguez. 2014. *Metodología de desarrollo para la actividad productiva de la.* La Habana : s.n., 2014.

32. Sánchez, Yasmany Aguilera. 2016. *Análisis y Diseño de nuevas funcionalidaddes para el módulo actividades en la herramienta de Autor Web Croda 2.0.* La Habana, Cuba : s.n., 2016.

33. Schmuller, J. 2000. *Aprendiendo UML en 24horas.* México : Pearson Educación, 2000.

34. Sommerville, Ian. 2005. *Ingeniería del software. Séptima edición.* Madrid : s.n., 2005.

35. —. *Requisitos de Software. Octava edición.*

36. STROUSTRUP, Bjarne. 1986. *The C++ programming language.* 1986.

GLOSARIO DE TÉRMINOS

CASE: las herramientas CASE (*Computer Aided Software Engineering*) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en término de tiempo y de dinero.

Framework: Marco de trabajo.

GNU: es un sistema operativo de tipo Unix desarrollado por y para el Proyecto GNU. GNU es un acrónimo recursivo de "GNU's Not Unix!" (En español: GNU no es Unix).

GNU/Linux: el núcleo Linux se complementa con una serie de aplicaciones desarrolladas por el grupo GNU para conformar el sistema operativo de *software* libre GNU/Linux.

GRASP: General Responsibility Assignment Software Patterns
GoF: Gang of Four, Pandilla de los Cuatro.

GB: Gigabyte, es una unidad de almacenamiento.
HDD: Unidad de Disco Duro: Hard Disk Drive.

RAM: Memoria de Acceso Aleatorio: Random Access Memory.

Sistemas: del latín systema, es un módulo ordenado de elementos que se encuentran interrelacionados y que interactúan entre sí.

UML: Lenguaje Unificado de Modelado (por sus siglas en inglés, Unified Modeling Language.)

Visualización: es el acto y la consecuencia de visualizar. Este verbo, por su parte, refiere a desarrollar mentalmente la imagen de algo abstracto, a otorgar características visibles a aquello que no se ve.

ANEXOS

Anexo 1: Historias Usuario.

Historia de Usuario	
Número: 2.	**Nombre:** Permitir la gestión del tipo de dato Enum.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Media
Puntos Estimados: 0.20	**Iteración asignada:**1
Puntos Reales:0.40	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato Enum.	
Observaciones:	

Tabla 8. HU2.

Historia de Usuario	
Número: 3.	**Nombre:** Permitir la gestión del tipo de dato Group.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Puntos Estimados: 0.20	**Iteración asignada:**1
Puntos Reales:0.60	
Descripción: La presente historia de usuario tiene como objetivo permitir agrupar a las propiedades por Grupos.	
Observaciones:	

Tabla 9. HU3.

Historia de Usuario	
Número: 4.	**Nombre:** Permitir la gestión del tipo de dato Int.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada:**1
Puntos Reales:0.80	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato Int.	
Observaciones:	

Tabla 10. HU4.

Historia de Usuario	
Número: 5.	**Nombre**: Permitir la gestión del tipo de dato QBrush.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada:** 1
Puntos Reales: 1	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato QBrush.	
Observaciones:	

Tabla 11. HU5.

Historia de Usuario	
Número: 6.	**Nombre**: Permitir la gestión del tipo de dato QPen**.**
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada: n**
Puntos Reales: 1.20	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato QPen.	
Observaciones:	

Tabla 12. HU6.

Historia de Usuario	
Número: 7.	**Nombre:** Filtrar propiedades por nombre
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.40	**Iteración asignada:** 1
Puntos Reales: 1.60	
Descripción: La presente historia de usuario tiene como objetivo permitir filtrar las propiedades por su nombre.	
Observaciones:	

Tabla 13. HU7.

Historia de Usuario	
Número: 8.	**Nombre**: Permitir la gestión del tipo de dato QDate.
Programador: Yohan Diaz Zayas	

Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada:** 1
Puntos Reales: 1.80	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato QDate.	
Observaciones:	

Tabla 14. HU8.

Historia de Usuario	
Número: 9.	**Nombre:** Permitir la gestión del tipo de dato QTime.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada:** 1
Puntos Reales: 2	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato QTime.	
Observaciones:	

Tabla 15. HU9.

Historia de Usuario	
Número: 10.	**Nombre:** Permitir la gestión del tipo de dato QDateTime.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada:** 1
Puntos Reales: 2.20	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato QDateTime.	
Observaciones:	

Tabla 16. HU10.

Historia de Usuario	
Número: 11.	**Nombre:** Permitir la gestión del tipo de dato String.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada:** 1
Puntos Reales: 2.40	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato String.	

Observaciones:

Tabla 17. HU11.

Historia de Usuario	
Número: 12.	**Nombre:** Permitir la gestión del tipo de dato QFon**t.**
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada:** 1
Puntos Reales: 2.60	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato QFont.	
Observaciones:	

Tabla 18. HU12.

Historia de Usuario	
Número: 13.	**Nombre:** Permitir la gestión del tipo de dato QColor.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 0.20	**Iteración asignada:** 1
Puntos Reales: 2.80	
Descripción: La presente historia de usuario tiene como objetivo permitir la visualización y modificación del tipo de dato QColor.	
Observaciones:	

Tabla 19. HU13.

Historia de Usuario	
Número: 14.	**Nombre:** Crear controlador para la manipulación de las propiedades del objeto.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 1	**Iteración asignada:** 2
Puntos Reales: 3.80	
Descripción: La presente historia de usuario tiene como objetivo permitir la creación controlador para la manipulación de las propiedades del objeto.	
Observaciones:	

Tabla 20. HU14.

Historia de Usuario	
Número: 15.	**Nombre:** Crear delegado para la renderización de las propiedades del objeto.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 1	**Iteración asignada:** 2
Puntos Reales: 4.80	
Descripción: La presente historia de usuario tiene como objetivo permitir la creación de delegado para la renderización de las propiedades del objeto.	
Observaciones:	

Tabla 21. HU15.

Historia de Usuario	
Número: 16.	**Nombre:** Crear modelo para las propiedades del objeto.
Programador: Yohan Diaz Zayas	
Prioridad en Negocio: Alta	**Prioridad en Negocio:** Alta
Puntos Estimados: 1	**Iteración asignada:** 2
Puntos Reales: 5.80	
Descripción: La presente historia de usuario tiene como objetivo permitir un modelo para las propiedades del objeto.	
Observaciones:	

Tabla 22. HU16.

Anexo 2: Prueba de Aceptación.

Caso de prueba de aceptación	
Código: HU2_P1	**Historia de usuario:2**
Nombre: Permitir la gestión del tipo de dato Enum.	
Descripción: El sistema debe visualizar el tipo de dato Enum, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha inferior se muestra tipo de dato Enum.	

Tabla 23. Prueba de aceptación#2.

Caso de prueba de aceptación	
Código: HU3_P1	**Historia de usuario:3**
Nombre: Permitir la gestión del tipo de dato Group.	
Descripción: El sistema debe visualizar el tipo de dato Group.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato Group.	

Tabla 24. Prueba de aceptación#3.

Caso de prueba de aceptación	
Código: HU4_P1	**Historia de usuario:4**
Nombre: Permitir la gestión del tipo de dato Int.	
Descripción: El sistema debe visualizar el tipo de dato Int, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato Int.	

Tabla 25. Prueba de aceptación#4.

Caso de prueba de aceptación	
Código: HU5_P1	**Historia de usuario:5**
Nombre: Permitir la gestión del tipo de dato QBrush.	

Descripción: El sistema debe visualizar el tipo de dato QBrush., permitiendo modificarlo.
Condición de ejecución:
Pasos de ejecución: El usuario ejecuta la aplicación.
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato QBrush.

Tabla 26. Prueba de aceptación#5.

Caso de prueba de aceptación	
Código: HU6_P1	**Historia de usuario:6**
Nombre: Permitir la gestión del tipo de dato QPen.	
Descripción: El sistema debe visualizar el tipo de dato QPen, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato QPen.	

Tabla 27. Prueba de aceptación#6.

Caso de prueba de aceptación	
Código: HU7_P1	**Historia de usuario:7**
Nombre: Permitir la gestión del tipo de dato QDate.	
Descripción: El sistema debe visualizar el tipo de dato QDate, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato QDate.	

Tabla 28. Prueba de aceptación#7.

Caso de prueba de aceptación	
Código: HU8_P1	**Historia de usuario:8**
Nombre: Permitir la gestión del tipo de dato QTime.	
Descripción: El sistema debe visualizar el tipo de dato QTime, permitiendo modificarlo.	
Condición de ejecución:	

Pasos de ejecución: El usuario ejecuta la aplicación.
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato QTime.

Tabla 29. Prueba de aceptación#8.

Caso de prueba de aceptación	
Código: HU9_P1	**Historia de usuario:9**
Nombre: Permitir la gestión del tipo de dato QDateTime.	
Descripción: El sistema debe visualizar el tipo de dato QDateTime, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato QDateTime.	

Tabla 30. Prueba de aceptación#9.

Caso de prueba de aceptación	
Código: HU10_P1	**Historia de usuario:10**
Nombre: Permitir la gestión del tipo de dato QDataTime.	
Descripción: El sistema debe visualizar el tipo de dato QDataTime, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato QDataTime.	

Tabla 31. Prueba de aceptación#10.

Caso de prueba de aceptación	
Código: HU11_P1	**Historia de usuario:11**
Nombre: Permitir la gestión del tipo de dato String.	
Descripción: El sistema debe visualizar el tipo de dato String, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato String.	

Tabla 32. Prueba de aceptación#11.

Caso de prueba de aceptación	
Código: HU12_P1	**Historia de usuario:12**
Nombre: Permitir la gestión del tipo de dato QFont.	
Descripción: El sistema debe visualizar el tipo de dato QFont, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato QFont.	

Tabla 33. Prueba de aceptación#12.

Caso de prueba de aceptación	
Código: HU13_P1	**Historia de usuario:13**
Nombre: Permitir la gestión del tipo de dato QColor.	
Descripción: El sistema debe visualizar el tipo de dato QColor, permitiendo modificarlo.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha se muestra tipo de dato QColor.	

Tabla 34. Prueba de aceptación#13.

Caso de prueba de aceptación	
Código: HU18_P1	**Historia de usuario:18**
Nombre: Filtrar propiedades por nombre.	
Descripción: El sistema debe visualizar y permitir filtrar una propiedad por su nombre.	
Condición de ejecución:	
Pasos de ejecución: El usuario ejecuta la aplicación.	
Resultado esperado: En la barra de estado que se encuentra en la esquina derecha inferior se muestra para filtrar una propiedad por su nombre.	

Tabla 35. Prueba de aceptación#14.

Anexo 3: Patrones GoF.

Creacionales	Estructurales	Comportamiento
Método de fabricación	Adaptador	Intérprete
Fábrica abstracta	Puente	Método plantilla
Constructor virtual	Objeto compuesto	Cadena de responsabilidades
Prototipo	Decorador	Orden
Instancia única	Fachada	Iterador
	Peso ligero	Mediador
	Proxy	Recuerdo
		Observador
		Estado
		Estrategia
		Visitante

Tabla 36. Patrones GoF.

Anexo 4: Inspector de propiedades

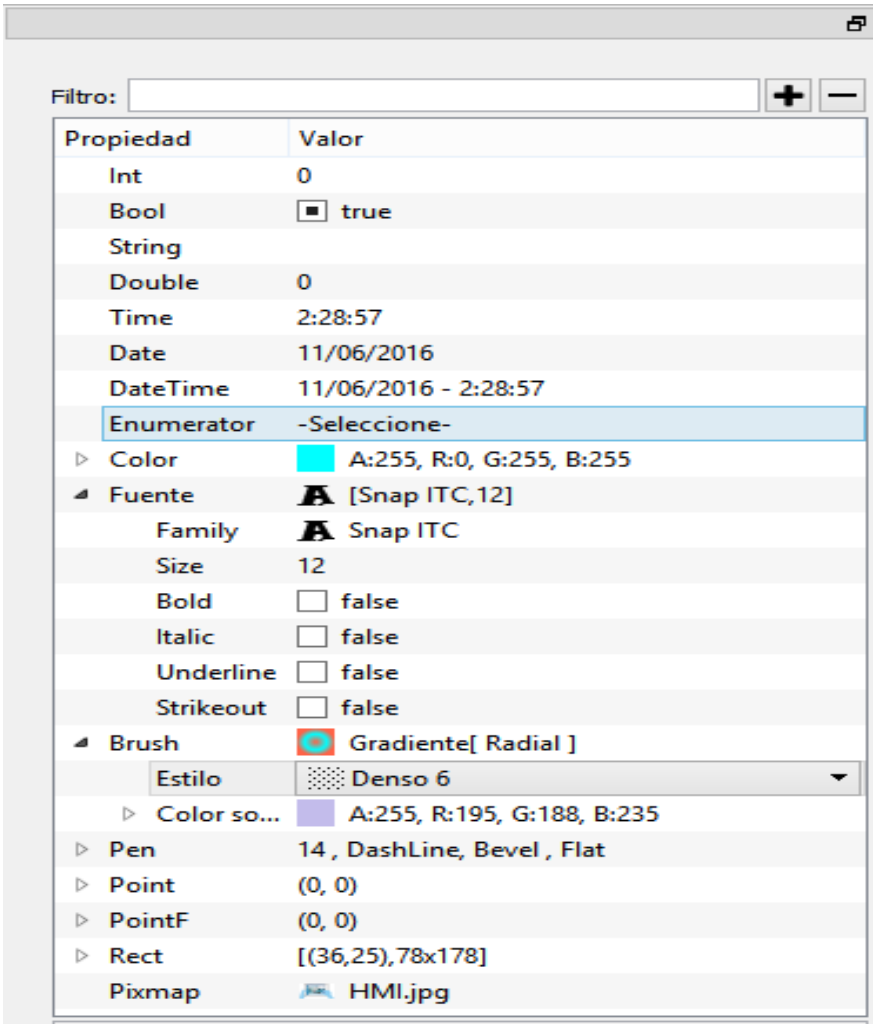


Figura 23. Inspector de propiedades de la solución.

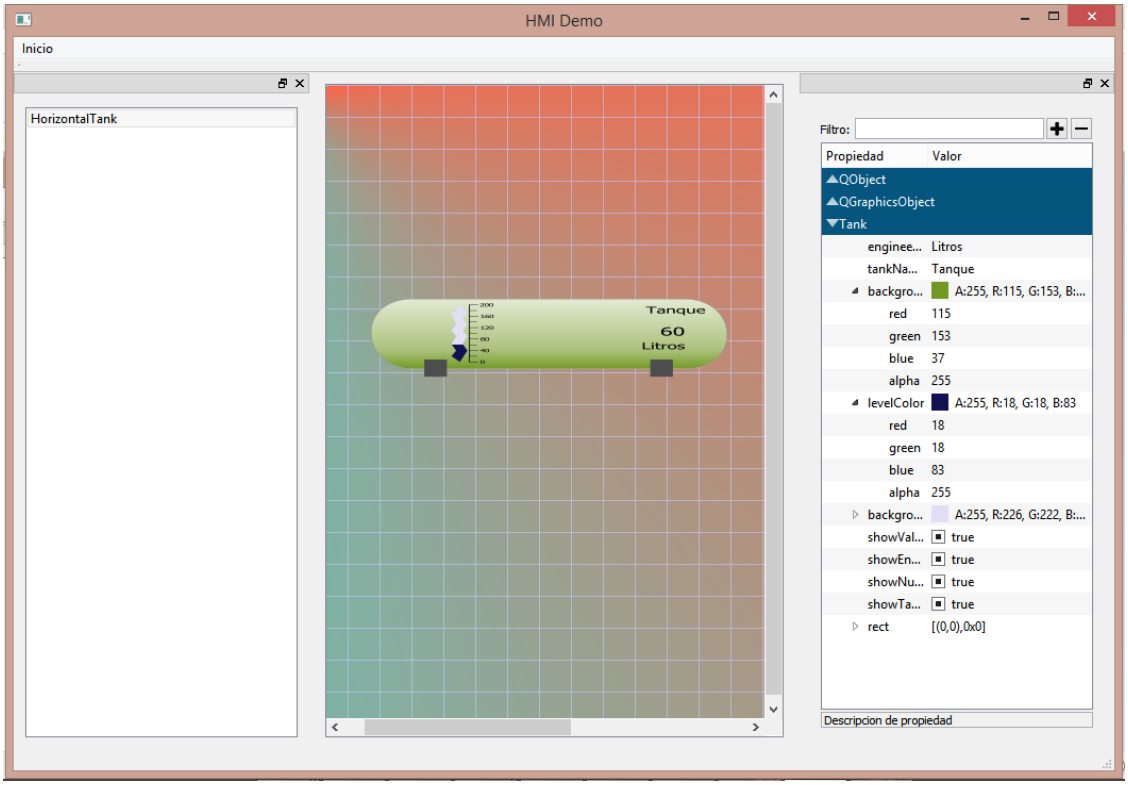


Figura 24. Inspector de propiedades, examinando un objeto.