Universidad de las Ciencias Informáticas

FACULTAD 6

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Framework de desarrollo basado en componentes sobre Qt en su v2.0

Autores: Ismaray Socarrás Ramírez

Eleanet Bujan Ponte

Tutores: Ing. Yusdenys Pérez Mendoza

Ing. Yunet Gasca Suárez

Consultante: Ing. Yunior Mesa Reyes

La Habana, junio 2014

"Año 56 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que _____ y _____ somos los únicos autores de este trabajo y autorizamos a 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 2014.

_____	_____
Firma del Autor	Firma del Autor
Ismaray Socarrás Ramírez	Eleanet Bujan Ponte
_____	_____
Firma del Tutor	Firma del Tutor
Ing. Yusdenys Pérez Mendoza	Ing. Yunet Gasca Suárez

DATOS DE CONTACTO

Tutor: Ing. Yusdenys Pérez Mendoza (ypmendoza@uci.cu)

Graduado de Ingeniero en Ciencias Informáticas en la UCI en el año 2009. Trabajador del centro GEYSED, en el cual se desempeña como Asesor de Calidad.

Tutora: Ing. Yunet Gasca Suárez (ygasca@uci.cu)

Graduada de Ingeniera en Ciencias Informáticas en la UCI en el año 2013. Trabajadora del centro GEYSED en el proyecto Calidad SD.

Autor: Ismaray Socarrás Ramírez (isocarras@estudiantes.uci.cu)

Estudiante perteneciente al departamento de Señales Digitales del centro GEYSED, específicamente al proyecto Calidad SD. Desempeña el rol de Analista.

Autor: Eleanet Bujan Ponte (ebujan@estudiantes.uci.cu)

Estudiante perteneciente al departamento de Señales Digitales del centro GEYSED, específicamente al proyecto Calidad SD. Desempeña el rol de Analista.

AGRADECIMIENTOS

Ismaray

A mis padres les agradezco en primer lugar este logro, por haberme dado la vida y el lugar que me corresponde. Por ser mis tesoros más preciado y estar siempre para mí en las buenas y en las malas.

A mi abuela Raquel por ser más que mi abuela mi madre, por quererme y hacerme sentir importante, por apoyarme a pesar de sus prejuicios, por perdonarme a pesar de mis errores.

A Haydee Tonso Manin (mi tata) por ser como una madre más para mí y estar siempre muy cerca de mí en los buenos y malos momentos de mi vida, aunque sea de lejos.

A Roexcy Vega Prieto por ayudarme desde el principio de carrera a que saliera adelante sin problemas y pudiera graduarme como lo que quería una ingeniera.

A mis tíos: Karenia y Roberto que obran por la vida sin interés de recibir nada cambio, por ser familiares y velar por el bienestar de todos.

A mi amiga y compañera de tesis Eleanet Bujan Ponte por estar a mi lado en las buenas y en las malas, gracias por ser mi amiga incondicional, por siempre decirme la verdad y no lo que quisiese oír, la verdad es que eres especial para mí y quiero que quede así para siempre.

AGRADECIMIENTOS

Eleanet

Agradezco antes que todo a mis adorados padres, por ser un ejemplo incondicional en mi vida. Por darme las fuerzas que se necesitan cuando estás lejos del hogar, y poder cumplir todas mis metas. Por educarme, cuidarme, quererme tanto y hacerme la persona que soy hoy.

A mis queridos abuelos Luis, Neormicia y Lázaro por ser mis segundos padres y brindarme todo su amor y cariño.

A toda mi familia por estar siempre presente dándome su apoyo en los momentos buenos y malos. Por darme consejos que me permiten ser una mejor persona.

A mi queridísima madrina Haydé, por quererme como su hija y tenerme en un lugar especial en su corazón, como yo la tengo en el mío.

A mis preciados suegros Lupe y Robertico por apoyarme siempre, brindarme toda su atención y confianza y hacerme sentir tan bien en su hogar.

A mi adorado novio Roberto por estar conmigo desde el comienzo de mi carrera dándome todo su apoyo, principalmente en este último año. Por ser una persona tan especial en mi vida y por darme todo su amor y cariño.

A todas mis amistades, compañeros de clase, tutores y profesores que me ayudaron a lo largo de la carrera, principalmente este último año.

A mi compañera de tesis Ismaray por convertirse en una buena amiga y apoyarme siempre que necesité su ayuda.

DEDICATORIA

A nuestros padres, hermanos, abuelos y a todos aquellos amigos que de una forma u otra han contribuido a que hoy hayamos alcanzado la meta trazada.

A nuestro Comandante en Jefe Fidel Castro Ruz, por habernos guiado hacia un futuro de hombres de ciencia, por haber forjado una Revolución donde todos tienen acceso a la educación, desde el más pobre hasta el que vive en el lugar más intrincado del país.

RESUMEN

El desarrollo de software basado en componentes se ha convertido en uno de los paradigmas de programación más efectivo, en el perfeccionamiento de aplicaciones complejas. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de un software basado en componentes mejorará con el paso del tiempo. Siendo consecuente con los beneficios intrínsecos de este paradigma, el departamento de Señales Digitales de la Universidad de las Ciencias Informáticas (UCI) tiene como prioridad potenciar la producción y desarrollo de software que fortalezcan estas ventajas.

La presente investigación tuvo como propósito desarrollar una nueva versión del Framework Basado en Componentes sobre Qt. Esta nueva herramienta fue desarrollada para el departamento Señales Digitales, garantizando que componentes cargados por varias aplicaciones pudiesen intercambiar información y comunicarse a través de la red. De esta forma se garantiza la arquitectura distribuida que poseen la mayoría de los proyectos del departamento. Además posibilita cargar componentes ActiveX y Kpart, los cuales proporcionan interfaces estándar para la comunicación con otros componentes y ofrecen un marco de reutilización de código.

El documento refleja todo el ciclo de desarrollo del software, respondiendo a las directivas que propone la metodología seleccionada para guiar todo el proceso. Se realiza un análisis y estudio, mediante métodos teóricos de algunos de los sistemas desarrollados siguiendo el paradigma de desarrollo de software basado en componentes. Se generan los artefactos principales propuestos para cada etapa; obteniendo como resultado una correcta implementación de la solución.

Palabras Claves: componentes, paradigma.

ABSTRACT

Component-based software development has become one of the most effective programming paradigms. As a component can be made and later continuously improved by an expert or organization, the quality of component-based software will improve with the passage of time. Taking into account the intrinsic benefits of this paradigm, the Digital Systems Department at the University of Informatics Sciences (UCI) has the priority of encouraging software development for strengthening these advantages.

The present research work aimed at developing a new version of the Component-based Framework on Qt. This new tool was developed by the Digital Systems Department, in a way that it guarantees that components loaded by various applications can exchange information and communicate through the network. In this way, distributed architecture of the projects at the department is guaranteed. Besides, it makes it possible to load the active components ActiveX and Kpart, which provide standard interfaces for communication with other components and offer a code reutilization framework.

The document presented shows the whole software development cycle, according to the regulations of the selected process methodology. An analysis and a study by theoretical methods of some of the developed systems are made, following the component-based paradigm. Main artifacts proposed for each stage and a correct programming of the solution is obtained.

KEY WORD: components, paradigm.

ÍNDICE

INTRODUCCIÓN **1**

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA ASOCIADA AL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES **5**

Introducción 5

1.1 Conceptos asociados al dominio del problema 5

Componente de software *5*

Interfaces *6*

ActiveX *6*

Kpart… 6

Reutilización de componentes *7*

Interoperabilidad *7*

1.2 Objeto de estudio 7

Descripción general del objeto de estudio 7

1.3 Situación problémica 9

1.4 Análisis de soluciones existentes 10

CORBA 10

GCF (Generic Component Framework) 11

Framework para el Desarrollo Basado en Componentes sobre Qt. 12

1.5 Conclusiones Parciales 13

CAPÍTULO 2: TECNOLOGÍAS Y HERRAMIENTAS A UTILIZAR **14**

Introducción 14

2.1 Metodología de desarrollo de software 14

Metodología Ágil. Programación Extrema (XP) 14

Fundamentación de la metodología seleccionada 15

2.2 Framework de desarrollo 15

Framework Qt v4.8 15

Fundamentación del framework de desarrollo seleccionado 16

2.3 Entorno de desarrollo Integrado (IDE), Qt Creator 2.4 16

Fundamentación del IDE seleccionado 16

ÍNDICE

2.4 Lenguaje de programación 16
Lenguaje de programación C ++ 17
Fundamentación del lenguaje de programación seleccionado 17
2.5 Herramienta CASE (Computer Aided Software Engineering) 17
Visual Paradigm for UML Enterprise Edition 8.0 17
Fundamentación de la herramienta CASE seleccionada 17
2.6 Lenguaje de Modelado 17
UML v2.0 18
Fundamentación del lenguaje de modelado seleccionado 18
2.7 Biblioteca 18
Libqxt18
2.8 Conclusiones Parciales 19
CAPÍTULO 3: CARACTERÍSTICAS DEL SISTEMA **20**
Introducción 20
3.1 Fase de Exploración 20
Historias de Usuario (HU) 21
Riesgo en su desarrollo 21
3.4 Fase de Planificación 22
3.5 Fase de Iteración 22
3.6 Diseño del sistema 22
Descripción del sistema propuesto 22
Diagrama de clases del diseño 23
Arquitectura del framework 24
Estilos Arquitectónicos 25
Patrones de Diseño 25
3.7 Artefactos generados 29
Lista de reserva del producto 30
Prioridad de las Historias de Usuario 32
Estimación de esfuerzo de las Historias de Usuario 32
Plan de entregas 33
Tareas de ingeniería o de programación por HU 34

ÍNDICE

Tarjetas CRC (Clase, Responsabilidad y Colaborador) 36
3.8 Conclusiones Parciales 37
CAPÍTULO 4. PRUEBAS DE SOFTWARE **38**
Introducción 38
4.1 Pruebas Unitarias 38
Pruebas de Caja Blanca 38
Paso 1: Dibujar el grafo de flujo asociado al método searchConnectionSentence. 38
Paso 2: Cálculo de la Complejidad Ciclomática (V (G)) 39
Paso 3: Determinar el conjunto básico de caminos independientes. 40
Paso 4: Casos de Pruebas 40
Grafo de flujo asociado al método connectComponents. 41
Cálculo de la Complejidad Ciclomática (V (G)) 42
Conjunto básico de caminos independientes: 42
Casos de Pruebas: 42
4.2 Pruebas funcionales 43
Pruebas de regresión 43
4.3 Conclusiones 45
CONCLUSIONES GENERALES **46**
RECOMENDACIONES **47**
BIBLIOGRAFÍA REFERENCIADA **48**
BIBLIOGRAFÍA CONSULTADA **51**
ANEXOS **54**
Anexo#1 Entrevistas 54
Anexo # 2. Tareas Ingenieriles 54
Anexo# 3: Numeración de los métodos escogidos para realizar el grafo de flujo. 58
Anexo#4: Casos de prueba 60
Anexo#5: Estándar de codificación utilizado 62

ÍNDICE DE TABLAS

Tabla 1: Personal relacionado con el framework 23
Tabla 2: Historia de Usuario Comunicar componentes a través de la red 29
Tabla 3: Historia de Usuario Cargar componente ActiveX.....30
Tabla 4: Historia de Usuario Cargar componente Kpart 30
Tabla 5: Lista de reserva del producto 31
Tabla 6: Prioridad de las Historias de Usuario 32
Tabla 7: Estimación del esfuerzo necesario por Historia de Usuario 32
Tabla 8: Plan de entrega de las historias de usuario 34
Tabla 9: Plan de iteraciones.....34
Tabla 10: Tarea de ingeniería # 3 de la HU Comunicar componentes de diferentes aplicaciones a través de la red. 35
Tabla 11: Tarea de ingeniería # 1 de las HU Cargar Componente ActiveX y Cargar Componente Kpart 35
Tabla 12: Tarjeta CRC #1 “Comunicación”.....36
Tabla 13: Tarjeta CRC #2 “Carga”.....36
Tabla 14: Tarjeta CRC #3 “XML” 36
Tabla 15: Caso de prueba para el camino básico No.6 de la funcionalidad searchConnectionSentence 40
Tabla 16: Caso de prueba para el camino básico No.8 de la funcionalidad searchConnectionSentence 41
Tabla 17: Caso de prueba para el camino básico No.9 de la funcionalidad connectComponents42
Tabla 18: Caso de prueba para el camino básico No.2 de la funcionalidad connectComponents43
Tabla 19: Caso de prueba asociado a la funcionalidad loadComponent 44
Tabla 20: Caso de prueba asociado a la funcionalidad searchComponent 44
Tabla 21: Caso de prueba asociado a la funcionalidad connectionExists.....44
Tabla 22: Tarea de ingeniería # 2 de la HU Cargar componente Kpart y ActiveX.....54
Tabla 23: Tarea de ingeniería # 3 de la HU Cargar componente Kpart y ActiveX.....55
Tabla 24: Tarea de ingeniería # 4 de la HU Cargar componente Kpart y ActiveX.....55

ÍNDICE DE TABLAS

Tabla 25: Tarea de ingeniería # 1 de la HU Comunicar componentes de diferentes aplicaciones a través de la red. 56
Tabla 26: Tarea de ingeniería # 2 de la HU Comunicar componentes de diferentes aplicaciones a través de la red. **.....** **……..**56
Tabla 27: Tarea de ingeniería # 4 de la HU Comunicar componentes de diferentes aplicaciones a través de la red. **.....**57
Tabla 28: Tarea de ingeniería # 5 de la HU Comunicar componentes de diferentes aplicaciones a través de la red. **.....**57
Tabla 29: Caso de prueba para el camino básico No.1 de la funcionalidad connectComponents **.....……..**60
Tabla 30: Caso de prueba para el camino básico No.4 de la funcionalidad connectComponents **.....……..**61
Tabla 31: Caso de prueba para el camino básico No.7 de la funcionalidad connectComponents .....……..61

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Fig. 1: Descripción del funcionamiento del Framework Basado en Componentes sobre Qt en su primera versión. 9
Fig. 2: Diagrama de Clases del Diseño asociado al framework basado en componentes 24
Fig. 3: Representación del patrón Experto en el Framework Basado en Componentes sobre Qt 26
Fig. 4: Representación del patrón Creador en el Framework Basado en Componentes sobre Qt 27
Fig. 5: Representación del patrón Controlador en el Framework Basado en Componentes sobre Qt 28
Fig. 6: Evidencia del uso del patrón MetaObject 28
Fig. 7: Representación del patrón Adapter en el Framework Basado en Componentes sobre Qt 29
Fig. 8: Grafo de flujo asociado al método searchConnectionSentence 39
Fig. 9: Grafo de flujo asociado al método connectComponents 41
Fig. 10: Numeración del método searchConnectionSentence 58
Fig. 11: Numeración del método connectComponents 59
Fig. 12: Funcionalidad loadComponent 59
Fig. 13: Funcionalidad searchComponent 60
Fig. 14: Funcionalidad connectionExists 60

INTRODUCCIÓN

La industria del software ha adquirido gran auge en las últimas décadas. Cada día son más las demandas de diferentes tipos de software y menor el tiempo disponible para desarrollarlos, cumpliendo siempre los estándares más altos de calidad. Para hacer frente a estas demandas los desarrolladores han promovido la reutilización de software ya existente, logrando construir sistemas cada vez más completos y diversos, con bases firmes y calidad incomparable. Para poder utilizar positivamente las ventajas implícitas de la reutilización de software, se concibió y perfeccionó la Ingeniería de Software Basada en Componentes (ISBC), la cual está centrada en el diseño y construcción de sistemas informáticos que utilizan componentes de software reutilizables.

Dentro de esta ingeniería está concebido el paradigma de Programación Basado en Componentes (PBC), que enfatiza en la descomposición de sistemas previamente conformados por componentes lógicos o funcionales. Son definidas además, un conjunto de interfaces que son utilizadas para la comunicación entre dichos componentes. La PBC constituye una de las respuestas a la creciente necesidad de desarrollar sistemas complejos en cortos períodos de tiempo, con los menores esfuerzos posibles, ya sean humanos o económicos.

La Universidad de las Ciencias Informáticas (UCI) cuenta con un Centro de Geoinformática y Señales Digitales (GEYSED). Dicho centro está conformado por dos departamentos. Señales Digitales es el encargado de producir software para el procesamiento de señales digitales tomando como referencia videos e imágenes, así como Sistemas de Transmisión de Televisión Digital, mientras que Geoinformática desarrolla Sistemas de Información Geográfica.

El proyecto Video Vigilancia (VV), perteneciente al departamento Señales Digitales, cuenta con un Framework[1] Basado en Componentes sobre Qt (FBC), implementado con el objetivo de acoplar distintos componentes de software desarrollados por los proyectos del departamento. Este FBC define la estructura, interrelación e integración de los componentes para los proyectos, solucionando la escasa comunicación que existía en las aplicaciones desarrolladas por diferentes proyectos, que de cierta forma eran dependientes. Gran parte de los proyectos que pertenecen al departamento están estructurados

[1] Conjunto cohesivo de interfaces y clases que colaboran para proporcionar los servicios de la parte central e invariable de un subsistema lógico

mediante una arquitectura distribuida[2], sin embargo el *framework* con que se cuenta para el trabajo con componentes, limita la comunicación a los componentes que él mismo define en su arquitectura, imposibilitando el intercambio de información con otros, ubicados en distintas estaciones de trabajo conectadas a la red.

Con el avance de la tecnología son desarrollados diversos componentes y controles que pueden ser integrados en software ya existente o ser utilizados en la implementación de uno nuevo. El Framework Basado en Componentes con que cuenta el departamento solo permite cargar componentes que tienen definida una estructura específica. Dicha estructura no es similar a la estructura interna de otros tipos de componentes que beneficiarían a los proyectos del departamento. El proyecto Video Vigilancia, es uno de los ejemplos donde se evidencia lo planteado anteriormente, pues utiliza cámaras ip para la video vigilancia, las cuales traen integrados componentes que visualizan el flujo continuo de imágenes que reciben. Estos componentes ayudarían en gran medida a que su equipo de trabajo pueda obtener la información que brindan estos y poder utilizarla en el desarrollo de sus aplicaciones.

Teniendo en cuenta la problemática antes descrita, se define el siguiente **problema de investigación:** ¿Cómo permitir la interoperabilidad con componentes externos en el Framework Basado en Componentes sobre Qt?

Para dar solución al problema propuesto se establece como **objeto de estudio:** Framework Basado en Componentes sobre Qt, delimitando como **campo de acción:** La interoperabilidad entre componentes externos y el Framework Basado en Componentes sobre Qt desarrollado en el departamento Señales Digitales.

El **objetivo general** de la investigación es: Desarrollar una nueva versión del Framework Basado en Componentes sobre Qt que posibilite la comunicación e intercambio de información con componentes externos.

Se definieron las siguientes **preguntas científicas**, a las cuales se le darán respuesta en el transcurso de la investigación.

[2] Sistema en el que el procesamiento de información se distribuye sobre varias computadoras en vez de estar confinado en una única máquina

¿Cuáles son las características que debe cumplir el Framework Basado en Componentes sobre Qt v2.0 para permitir la interoperabilidad con componentes externos?

¿Cómo son las interrelaciones que se establecen en el Framework Basado en Componentes sobre Qt que permiten la comunicación entre componentes?

¿Cómo contribuir a que el Framework Basado en Componentes sobre Qt permita la interoperabilidad con componentes externos distribuidos en la red?

¿La nueva versión del *framework* posibilitó dar solución a la problemática existente?

Para dar cumplimiento al objetivo general, se trazaron las siguientes **tareas de investigación:**

- Caracterización de framework con características y funcionalidades similares al que se desea complementar.
- Selección y caracterización de las tecnologías y herramientas a utilizar para el desarrollo de la solución.
- Diseño de los artefactos definidos en las fases de exploración y planificación.
- Implementación de la solución.
- Descripción y aplicación de pruebas al Framework basado en componentes sobre Qt.

Con el propósito de dar cumplimiento al objetivo trazado, se lleva a cabo una investigación en la que se utilizaron métodos teóricos y técnicas de recolección de información.

Métodos Teóricos:

Analítico-Sintético: se utilizó para realizar un análisis de documentos y bibliografías de diferentes autores, obteniendo los elementos más importantes relacionados al desarrollo de software basado en componentes para la comunicación y carga de componentes de tipo ActiveX y Kpart en el Framework Basado en Componentes sobre Qt.

Histórico-Lógico: se empleó para analizar la evolución histórica de soluciones existentes similares al Framework Basado en Componentes sobre Qt a complementar, en cuanto a su fundamento sobre el paradigma de programación basado en componentes. Se realizó un estudio de su estado actual en

función de que las características que se ajustaran a las necesidades del proyecto pudieran ser utilizadas para el desarrollo en curso.

Técnica de recolección de información:

Entrevista: se empleó con el propósito de establecer diálogos de carácter investigativo con desarrolladores de la versión actual del Framework Basado en Componentes sobre Qt y con el Jefe de proyecto de Video Vigilancia Reynier Pupo Gómez. A través de estas se obtuvo información referente a la codificación del framework, el aporte de la nueva investigación y los requisitos funcionales a cumplir por la nueva versión del Framework Basado en Componentes sobre Qt. El tipo de entrevista utilizado fue la entrevista no estructurada, dado que el entrevistador elaboró las preguntas antes de realizar la misma, modificando el orden, la forma de encauzar las preguntas o su formulación para adaptarlas a las diversas situaciones y características particulares de los sujetos de estudio. **(Ver Anexo#1)**

Estructura de la investigación:

La investigación estará estructurada por: Resumen, Introducción, 4 Capítulos, Conclusiones, Recomendaciones, Bibliografía Referenciada, Bibliografía Consultada y Anexos. En el Capítulo #1 se explican los conceptos asociados al dominio del problema y se realiza un estudio del estado del arte referente al desarrollo de software basado en componentes. Luego en el Capítulo # 2 se fundamentará el uso de la metodología, tecnologías y herramientas sobre la cual se apoya la propuesta. Posteriormente en el Capítulo #3 se especificarán los requisitos funcionales y no funcionales que debe cumplir el FBC. Se describen además los estilos y patrones arquitectónicos y de diseño a utilizar. Finalmente en el Capítulo # 4: se abordará todo el proceso de construcción, selección y aplicación de pruebas a la solución final.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA ASOCIADA AL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES

Introducción

Como parte del proceso de investigación es necesario conocer los aspectos más importantes enmarcados en el objeto de estudio; para la comprensión tanto del equipo de desarrollo como de los usuarios. En este capítulo se abordan los principales elementos asociados al desarrollo de software basado en componentes, específicamente la carga de componentes de tipo ActiveX y Kpart y la comunicación entre componentes cargados por distintas aplicaciones distribuidas en la red, en el Framework Basado en Componentes sobre Qt, definiendo sus conceptos y terminologías. Se realiza una investigación sobre la situación actual del dominio del problema, así como de los procesos necesarios para la gestión de componentes, en este caso para los de tipo ActiveX y Kpart. Posteriormente se describen aplicaciones existentes con el propósito de tomar ideas o funcionalidades que sirven como punto de partida para desarrollar la solución.

1.1 Conceptos asociados al dominio del problema

Con el objetivo de lograr un mayor entendimiento del proceso que se lleva a cabo con respecto al desarrollo de software basado en componentes, se definieron los siguientes conceptos:

Componente de software

Según Szyperski un componente, *"es un paquete coherente de artefactos de software que puede ser desarrollado independientemente y entregado como una unidad y este puede ser compuesto, intercambiado con otro componente para construir algo mucho más grande".* (Szyperski, 2002)

Krutchen propone otra definición, la cual plantea que, *"un componente es una parte no trivial, casi independiente y reemplazable de un sistema que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. Un componente se conforma y provee la realización física por medio de un conjunto de interfaces."* (Krutchen, 2000)

Estos pueden ser utilizados en aplicaciones ya existentes o ser reutilizados en la implementación de nuevas soluciones. Pueden realizar funcionalidades específicas o interrelacionarse con otros componentes a través de una **interfaz**.

Interfaces

Según Palomino una interfaz, *"define el límite de comunicación entre dos elementos, tales como software-hardware o un usuario. Generalmente se refiere a una abstracción que un elemento provee de sí mismo al exterior. Esto separa los métodos de comunicación externa de los de operación interna, y le permite ser internamente modificada sin afectar la manera en que los elementos externos interactúan con él, también provee abstracciones múltiples de sí mismo."* (Palomino González, 2009)

Permiten exponer las funcionalidades de un componente específico y facilitan un mecanismo para interconectar los componentes y controlar las dependencias que generan. En el departamento Señales Digitales fue desarrollada la primera versión de un *framework,* descrito en la tesis de pregrado Framework para el desarrollo basado en componente sobre Qt. (Borges, 2013)

El mismo define las interrelaciones entre los componentes mediante la interfaz AbstractComponent, que es la que determina las funcionalidades de los componentes utilizados en el *framework.* Con el desarrollo de la segunda versión del *framework* los componentes del mismo podrán interoperar con otros entornos de componentes como **ActiveX** y **Kpart,** además de otros distribuidos en la red.

ActiveX

Un componente ActiveX es esencialmente *"Un objeto OLE (Object Linking and Embedding)[3] simple que admite la interfaz IUnknown. Por lo general, es compatible con muchas más interfaces para ofrecer funcionalidad, pero todas las interfaces adicionales se pueden ver como opcionales y, como tal, un contenedor no debe confiar en cualquier interfaz adicional que se admite".* (Microsoft, 2008)

Proporciona un entorno unificado de servicios que se basan en objetos con la capacidad de personalizar servicios y de ampliar arbitrariamente la arquitectura, permite la integración entre los componentes y los objetos, no necesitan conocer por anticipado con qué objetos se van a comunicar, ni su código necesita estar escrito en el mismo lenguaje. Además no se precisa de ningún código fuente, pues como el código original se ha convertido en un control ActiveX, es posible utilizarlo sin el apoyo de un programa compatible con ActiveX, ofrece un marco de reutilización de código, ya que son independientes del lenguaje.

Kpart

Kpart**, "***es el nombre de uno de los componentes para el entorno de escritorio KDE (K Desktop Environment). Pueden ser divididos en dos tipos: componentes y plugins. A un componente individual se*

[3] Estándar de vinculación e incrustación de objetos

le denomina un Kpart. Se fundamenta en bibliotecas compartidas, que hacen que el componente aparezca directamente como un objeto C+ +." (IBM, 2012)

Es un sistema de componentes embebidos y reutilizables que posee bibliotecas rápidas para activar o descargar componentes. Puede mostrar y extender la aplicación de menú para añadir las acciones específicas del componente y define los menús de una aplicación en un archivo XML (Lenguaje de Marcas Extensible).

Reutilización de componentes

Según Somerville la reutilización, *"es un enfoque de desarrollo que trata de maximizar el uso recurrente de componentes de software existentes."* (Sommerville, 2000)

Por otro lado Sametinger define como reutilización, *"al proceso de crear sistemas de software a partir de software existente, en lugar de desarrollarlo desde el comienzo."* (Sametinger, 1997)

Con el desarrollo de esta nueva solución los proyectos del centro GEYSED contarán con una herramienta para la creación de nuevas aplicaciones, reutilizando componentes ya existentes, sin tener que desarrollarlos desde un inicio.

Interoperabilidad

Según el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) interoperabilidad es: "*la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada"*. (Institute of Electrical and Electronics Engineers. IEEE , 1990)

La interoperabilidad es la capacidad que tiene un producto o un sistema, cuyas interfaces son totalmente conocidas, para funcionar con otros productos o sistemas existentes. (Dueñas Gómez, 2007)

La interoperabilidad en la primera versión del Framework Basado en Componentes sobre Qt es ineficiente, pues la comunicación que se establece entre los componentes del framework es limitada, imposibilitando que componentes ubicados en distintas aplicaciones, dentro del mismo ordenador o fuera de este, puedan intercambiar información entre ellos. La nueva versión del *framework* dará solución a esta limitante.

1.2 Objeto de estudio

Descripción general del objeto de estudio

Los *framework* son de vital importancia para construir grandes sistemas de software. Facilitan la programación de aplicaciones, encapsulando operaciones complejas en instrucciones sencillas. Si se usan de forma adecuada el software puede responder fácilmente a los cambios en los requerimientos

tanto del negocio como de tecnología. Los *framework* basados en componentes, son un acercamiento fundamentado en la reutilización para definir, implementar, y utilizar componentes débilmente acoplados en sistemas. Dichos componentes pueden producir o consumir eventos. Todos los procesos del sistema son colocados en componentes separados de tal manera que todos los datos y funciones dentro de cada componente están semánticamente relacionados. (Casal Terreros, 2014)

El Framework Basado en Componentes sobre Qt cuenta con dos partes fundamentales, la primera es la encargada de la creación de componentes y la segunda de crear una aplicación para la gestión de los mismos. Dichas partes están estrechamente relacionadas en función de visualizar las diferentes acciones que realizan los componentes. Las aplicaciones creadas por un usuario son capaces de cargar y comunicar componentes siempre y cuando estos posean la estructura de un *AbstractComponent*, estructura definida para los componentes que el *framework* posibilita cargar. Para ello cuenta con un archivo XML donde se especifica el nombre de los componentes que serán cargados. Además, define áreas de trabajo para la ubicación de componentes que posean una interfaz gráfica. En la raíz de dichas aplicaciones deben existir dos carpetas donde se ubicarán los componente que serán cargados y su archivo XML respectivamente.

Cada componente cuenta con un archivo XML donde se especifica su nombre y su ubicación en el área de trabajo. Además, en caso de que reciba alguna señal emitida por otro componente se especifica el nombre del emisor, la señal que emite, el nombre del receptor y el evento que realiza. Sin embargo, el *framework* no define las relaciones que se pueden establecer cuando un componente, dentro de la aplicación, necesita acceder a otro componente ubicado en otra estación de trabajo a través de enlaces de red. Para esto son empleadas técnicas tales como la serialización[4] (*marshalling*) para enviar la información brindada por un componente a su destino.

[4] La **serialización** es un mecanismo ampliamente usado para transportar objetos a través de una red, para hacer persistente un objeto en un archivo o base de datos, o para distribuir objetos idénticos a varias aplicaciones o localizaciones.

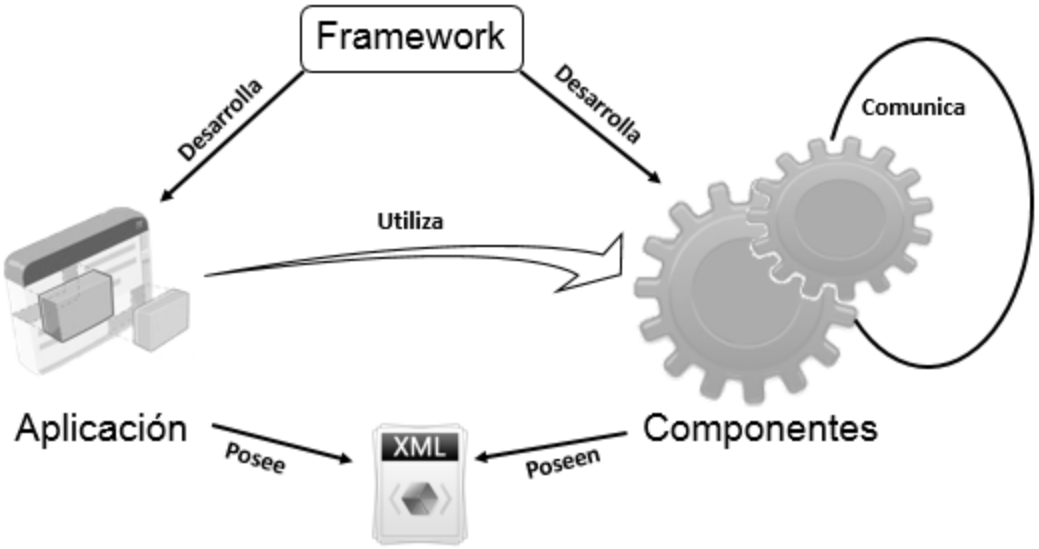

Fig. 1: Descripción del funcionamiento del Framework Basado en Componentes sobre Qt en su primera versión.

1.3 Situación problémica

El proyecto Video Vigilancia del departamento Señales Digitales desarrolla aplicaciones para la comercialización, además de otras que dan solución a necesidades específicas del centro GEYSED y del propio proyecto. Un ejemplo de lo expuesto con anterioridad es el Framework Basado en Componentes sobre Qt, el cual permite construir sistemas donde los desarrolladores tienen la libertad de agregar, separar y cambiar componentes, en función de cumplir con los requisitos definidos por el centro.

Este *framework* define la estructura que deben tener los componentes así como la aplicación que los carga utilizando APIs[5]. Para establecer la forma en que se cargarán los componentes y la manera en la que deben interactuar con otros, se definen reglas en archivos XML que acompañan a cada componente y a la aplicación. Una de las limitaciones que posee el *framework* es que las interrelaciones que se establecen entre sus componentes, no garantiza la comunicación con componentes cargados por otras aplicaciones. Esta limitación no permite aplicar una de las características principales de la arquitectura distribuida, la accesibilidad a la información, arquitectura que posee gran parte de los proyectos del

[5] Interfaz de programación de aplicaciones, que representa la capacidad de comunicación entre componentes de software.

departamento. El *framework* no es capaz de cargar componentes de tipo ActiveX y Kpart, los cuales son muy utilizados en diversos programas. Mediante estos componentes y controles se podrían manipular con mayor facilidad la información que brindan los programas integrados a las cámaras de video vigilancia. Además los proyectos del departamento podrán utilizarlos en función de sus necesidades.

1.4 Análisis de soluciones existentes

En la actualidad existen distintas herramientas que poseen funcionalidades similares a las que se le desean implementar a la nueva versión del Framework Basado en Componentes sobre Qt, por lo que se realizó una investigación sobre framework de desarrollo y estándares que operan sobre el paradigma de programación basado en componentes, con el propósito de realizar un análisis de las mismas. Estas herramientas pueden formar parte de una propuesta de solución o en otro caso hacer uso de sus características y funcionalidades para integrarlas a la nueva solución.

CORBA

CORBA es el acrónimo de *Common Object Request Broker Architecture*, estándar definido por *Object Management Group* (OMG) que permite que diversos componentes de software escritos en múltiples lenguajes de programación y que se ejecutan en diferentes computadoras, puedan trabajar juntos. Para cada tipo de objeto se define una interfaz IDL[6], la cual debe utilizar cualquier cliente que desea invocar una operación sobre el objeto, para especificar la operación que quiere llevar a cabo, y para calcular las referencias de los argumentos que se envían. Cuando la invocación alcanza el objeto de destino, la misma definición de interfaz se utiliza para deserializar los argumentos de manera que el objeto pueda realizar la operación solicitada. (Corba,2014)

La manera de realizar la invocación por parte del cliente es usando *stub*[7], una interfaz de comunicación con el servidor generada a partir del IDL. El cliente obtiene una referencia de uno o más objetos remotos en el servidor e invoca a sus métodos, sin tener que saber dónde se ejecuta el objeto, y a su vez el objeto no necesita saber desde donde se ejecuta el cliente. La noción de CORBA en cuanto a transparencia ha sido motivo de crítica, y esto es debido a que los objetos que residen en el mismo espacio de direcciones y que son accesibles con una simple llamada a una función son tratados como objetos que residen en

[6] Interface description language (también interface definition language)Describe una interfaz en un lenguaje neutral, lo cual permite la comunicación entre componentes de software desarrollados en diferentes lenguajes

[7] Simula el comportamiento de código existente (tal como un procedimiento en una máquina remota) o se comporta como el sustituto temporal para un código aún no desarrollado

cualquier parte del sistema distribuido. Esto hace que los accesos locales sean tan complicados como serían en el escenario remoto más complejo. (Corba,2014)

A pesar de no ser un *framework* se realizó una investigación sobre CORBA, ya que posibilita la interoperabilidad con componentes remotos y es un estándar que opera sobre el paradigma de programación basado en componentes. Se comprueba que no puede ser utilizado como vía de solución, pues los componentes que van a interactuar con la nueva versión del Framework Basado en Componentes sobre Qt no estarán localizados en un servidor, sino que estarán ubicados en aplicaciones distribuidas en distintos ordenadores.

GCF (Generic Component Framework)

Es una biblioteca de marco de trabajo para Qt que permite la creación de aplicaciones de software altamente extensibles y mantenibles. En el ámbito de GCF los plugins son llamados componentes, representados como un archivo de biblioteca dinámica. El elemento principal de una aplicación de este tipo es que ofrece una aplicación contenedora de *widget* dentro del cual se pueden mezclar los elementos de los componentes, ya sean elementos de menú o de barra de herramientas. Posibilita cargar componentes de tipo ActiveX y Kpart. Permite a los desarrolladores comunicarse con otras aplicaciones GCF por medio de señales y ranuras. Una aplicación GCF debe ir acompañada de un archivo XML, el cual contiene información acerca de los componentes que deben ser cargados por la aplicación. También contiene los principales parámetros de configuración para el software y la IPC[8] . (VCreate Logic, 2012)

La columna vertebral de dicho *framework* la componen tres elementos fundamentales:

- **La clase AbstractComponent**: de esta clase heredan los componentes que se van a desarrollar por el usuario, la misma garantiza que dichos componentes tengan una misma estructura, proporcionado una serie de datos básicos como el nombre de un componente y la ubicación del mismo. GCF ofrece un conjunto de servicios como por ejemplo: combinar la interfaz gráfica de usuario: acciones, menús, barras de herramientas, *widgets* y objetos del componente en la ventana principal de la aplicación. Decidir cómo los *widgets* de otros componentes se pueden colocar, quitar, estar oculto o mostrar en los *widgets* de este componente. Proporciona medios a través de los cuales el componente puede explorar objetos proporcionados por otros componentes en el sistema.

[8] Comunicación entre procesos (interprocess communication)

- **Archivo XML:** es un archivo fundamental para GCF. En él se describen los objetos, acciones, menús y barras de herramientas que expone un componente. El mismo recoge la forma en que se relacionarán los componentes entre ellos y la forma en que serán cargados. Todo componente GCF debe tener un archivo XML.
- **La clase Application:** de esta clase heredan las aplicaciones creadas por el usuario. Posee un archivo XML donde se determina los componentes que se ejecutarán.

El estudio realizado sobre la biblioteca GCF indica que aunque posee funcionalidades que pueden ser utilizadas en la solución, como comunicar componentes cargados por diferentes aplicaciones y cargar componentes de tipo ActiveX y Kpart, no puede ser adaptada a las necesidades del departamento pues este se rige por la licencia LGPL (*Lesser GPL*), que debido a sus beneficios comerciales puede utilizarse junto a software privativos en el desarrollo de nuevas aplicaciones. Mientras que GCF se desarrolla bajo la licencia GNU/GPL[9], la cual determina que si alguna de estas aplicaciones utiliza algún componente GPL, debe ser licenciado bajo la misma, es decir no se pueden utilizar partes o bibliotecas de software GPL en un software propietario o distribuido bajo otra licencia. Además exige que si se comercializan nuevas versiones de software, el código fuente debe estar a disposición de los usuarios.

Framework para el Desarrollo Basado en Componentes sobre Qt.

La primera versión de este *framework* fue creada en el proyecto Video Vigilancia, del departamento Señales Digitales del centro de desarrollo GEYSED. Cuenta con dos partes fundamentales, la primera es la encargada de la creación de componentes y la segunda de crear un entorno de trabajo para la gestión de los mismos. Cada componente cuenta con un archivo XML donde se especifica el nombre del componente y la ubicación del mismo en caso de poseer una interfaz gráfica de usuario. Además, en caso de que reciba alguna señal emitida por otro componente se especifica el nombre del emisor, la señal que emite, el nombre del receptor y el evento que realiza.

Las aplicaciones creadas por el usuario son capaces de cargar y comunicar componentes, para ello cuenta con un archivo XML donde se especifica el nombre de los componentes que se desean cargar. Además, cuenta con áreas definidas para la ubicación de componentes que poseen una interfaz gráfica de usuario. En la raíz de dichas aplicaciones existen dos carpetas que permiten la copia del componente que se desea cargar y su archivo XML respectivamente. (Borges, 2013.)

[9] *General Public License* (Licencia Pública General)

Finalmente se concluye que el Framework Basado en Componentes sobre Qt se tomará como punto de partida para el desarrollo de la investigación. Aunque no posibilita la comunicación entre componentes distribuidos por la red ni cargar componentes ActiveX o Kpart, haciendo uso del mismo no sería necesario implementar un nuevo framework, sino definirle nuevas funcionalidades que den solución al objetivo trazado.

1.5 Conclusiones Parciales

Las herramientas estudiadas aportaron ideas necesarias para dar solución a la problemática existente en el departamento. Aunque estas poseen características que pueden ser utilizadas para el desarrollo de la solución, se concluye que la primera versión del Framework Basado en Componentes sobre Qt, con nuevas funcionalidades a complementar, es el indicado para dar solución a la situación antes descrita, por la forma en que define los componentes y las reglas que establece para que interactúen entre sí.

CAPÍTULO 2: TECNOLOGÍAS Y HERRAMIENTAS A UTILIZAR

Introducción

Una de las primeras tareas a ejecutar, antes de implementar la solución que se propone, es realizar una investigación de las tendencias actuales, tecnologías existentes y selección de herramientas que contribuyan al desarrollo de aplicaciones de escritorio, dado que permiten guiar el proceso de desarrollo de software. Es por ello que en el presente capítulo se describen los aspectos relacionados a las directrices y tecnologías actuales a desarrollar en la producción de software, enfatizando en aquellas que se utilizan para el desarrollo del *Framework* Basado en Componentes sobre QT en su v2.0. Los aspectos mencionados están determinados por: la metodología para guiar el desarrollo del sistema de software, el *framework*, entorno de desarrollo y el lenguaje de programación.

2.1 Metodología de desarrollo de software

Una metodología de desarrollo de software es un conjunto de pasos o procedimientos que guían el proceso de construcción de un software, permitiendo obtener un producto con calidad y en el tiempo establecido. Según Marcos, *"una metodología de desarrollo debe tomarse como una guía, pero no como algo rígido; debe adaptarse para cada utilización de la misma, del mismo modo que el método de investigación.", también asegura que, "al iniciar un desarrollo de software se decide el paradigma metodológico (estructurado, orientado a objetos) y la metodología concreta a seguir".* (Marcos, 2005)

Metodología Ágil. Programación Extrema (XP)

Entre las metodologías ágiles se destaca la Programación Extrema[10], la cual está centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. (Beck, 1999)

La metodología de desarrollo XP está enfocada fundamentalmente en pruebas ya que estas constituyen una acción más que necesaria en cada iteración, permite que se prevean errores a medida que se programe. Según (Somerville, 2005): *"En la programación extrema todos los requerimientos se expresan como escenarios llamados historias de usuario, los cuales se implementan directamente como una serie*

[10] (XP), en inglés Extreme Programming

de tareas. Los programadores trabajan en parejas y desarrollan pruebas para cada tarea antes de escribir el código. Todas las pruebas se deben ejecutar satisfactoriamente cuando el código nuevo se integre al sistema. Existe un pequeño espacio de tiempo entre las entregas del sistema."

Fundamentación de la metodología seleccionada

Se hará uso de la metodología XP, ya que el equipo de desarrollo responde a una de sus características principales, la programación en parejas, lo que permite que mientras uno implemente, el otro corrija los errores que se van detectando y asegura que cuanto más grande se haga el proyecto, todo el equipo conocerá mucho mejor el sistema completo. Además es una metodología que al estar guiada por un proceso de pruebas, al culminar cada iteración permite identificar fallos, debidos a cambios recientes en el código. La misma permite que el cliente o un representante del cliente esté presente en todo momento del desarrollo de software para guiar al equipo de desarrollo, minimizando el riesgo de rehacer partes que no cumplen con los requisitos especificados por este y beneficiando a los programadores a centrarse en lo que es más importante. Otro elemento que se tuvo en consideración para la selección de esta metodología, es que la versión actual del Framework Basado en Componentes sobre Qt se desarrolló haciendo uso de la misma, lo que permite seguir una misma línea de trabajo y obtener una retroalimentación por parte del equipo de desarrollo para la nueva versión a realizar.

2.2 Framework de desarrollo

En el desarrollo de software, un *framework* es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas y bibliotecas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. (Riehle, 2000)

Framework Qt v4.8

Se caracteriza por ser un *framework* multiplataforma y orientado a objetos. Entre sus funciones más conocidas está la creación de interfaces de usuarios. Provee además diversas clases que facilitan el manejo de sockets[11], soporte para programación multihilo, comunicación con bases de datos, manejo de cadenas de caracteres. Brinda soporte para un elevado número de lenguajes de programación. (Martínez, 2011)

[11] Es un método para la comunicación entre un programa cliente y uno servidor en una red, por lo que se define como el punto final en una conexión.

Fundamentación del framework de desarrollo seleccionado

Se utilizará el Framework Qt en su versión 4.8, pues es el *framework* que se utiliza en gran parte de los proyectos del departamento, y al ser multiplataforma permite que una aplicación pueda ser compilada y utilizada en cualquier sistema operativo sin necesidad de cambiar el código continuamente. Otra de sus ventajas es que una de sus licencias es la LGPL, la cual posibilita que se utilice gratuitamente con fines comerciales. Además que la versión actual del Framework Basado en Componentes sobre Qt fue desarrollada haciendo uso del mismo.

2.3 Entorno de desarrollo Integrado (IDE), Qt Creator 2.4

Es un programa multiplataforma compuesto por una serie de herramientas que utilizan los desarrolladores para escribir el código. Está diseñado para el desarrollo de aplicaciones del framework Qt. Puede soportar varios lenguajes de programación o puede estar diseñado únicamente para uno. (Viñolo, 2012)

Fundamentación del IDE seleccionado

Se empleará como IDE Qt Creator en su versión 2.4, debido a que está orientado al desarrollo de aplicaciones a través del *framework* Qt, por lo que propicia una mayor integración y aumenta el número de funcionalidades. Además ofrece gran compatibilidad con varios sistemas operativos, posee un gran número de bibliotecas como QtCore, QtNetwork, QtGui, QtXML, entre otras, que permiten la obtención de nuevas aplicaciones de software.

2.4 Lenguaje de programación

Lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo o máquina debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo determinado. El lenguaje utilizado por el procesador se denomina lenguaje máquina, el cual trata los datos tal como llegan al procesador, en una serie de 0 y 1 o datos binarios. El lenguaje máquina, por lo tanto, no es comprensible para los seres humanos, razón por la cual se han desarrollado lenguajes intermediarios comprensibles para el hombre (lenguajes de alto nivel). El código escrito en este tipo de lenguaje se transforma en código máquina para que el procesador pueda procesarlo, consta de instrucciones independientes de la máquina; ha de ser compilado o interpretado para traducir su código en otro de bajo nivel. Por lo que constituye el lenguaje de programación más próximo a los usuarios en el proceso de desarrollo de software. (Guérin, 2011)

Lenguaje de programación C ++

Desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Se suele decir que C++ es un lenguaje de programación multiparadigma pues se unificó a paradigmas (programación estructurada y la programación orientada a objetos). Se caracteriza por una gran versatilidad y es potente frente al desarrollo de sistemas complejos. (Eckel, 2004)

Fundamentación del lenguaje de programación seleccionado

El análisis de la investigación sobre los diferentes tipos de lenguajes de programación existentes, concluye que el lenguaje que responde a las especificidades de la solución a desarrollar es C++, estándar de codificación ANSI C++98, por ser el lenguaje nativo del *framework* Qt.

2.5 Herramienta CASE (Computer Aided Software Engineering)

Las Herramientas CASE son definidas como herramientas individuales para ayudar al desarrollo de software o al administrador de proyecto durante una o más fases del desarrollo de software**.** Por lo que se define que esencialmente, CASE es una herramienta que ayuda al ingeniero de software a desarrollar y mantener software. (McClure, 1989.)

Visual Paradigm for UML Enterprise Edition 8.0

Visual Paradigm for UML ha sido concebido para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagrama, y la generación de código y documentación de los mismos. Permite el uso de un lenguaje estándar para todo el equipo de desarrollo lo que facilita la comunicación. (Gonzalo Génova Fuster, 2013)

Fundamentación de la herramienta CASE seleccionada

Se decidió el uso de esta herramienta para el diseño de las clases que conforman al framework. Además de que posee disponibilidad en múltiples plataformas y contribuye en lograr una mayor rapidez en el desarrollo del software, su licencia es gratuita. Al mismo tiempo el conocimiento que poseen los desarrolladores del equipo de trabajo de la herramienta es amplio, lo que facilita el diseño de los diagramas y la creación de los modelos necesarios que contribuyen a lograr la construcción de aplicaciones informáticas.

2.6 Lenguaje de Modelado

El lenguaje de modelado no es más que un lenguaje basado en diagramas para la especificación, visualización, construcción y documentación de cualquier sistema complejo. Captura decisiones y

conocimientos sobre los sistemas que se deben construir. Básicamente, un modelo es una simplificación de la realidad que se construye para comprender mejor el sistema que queremos desarrollar. (Larman, 2003)

UML v2.0

Lenguaje de Modelado Unificado es el lenguaje de modelado de sistemas de software para visualizar, especificar, construir y documentar un sistema. Ofrece además, un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Además de que los artefactos pueden tener características u operaciones asociadas, y pueden ser instanciados o asociados con otros artefactos. (James Rumbaugh, 1998)

Fundamentación del lenguaje de modelado seleccionado

Se decidió utilizar UML para el desarrollo de los diagramas correspondientes a la solución propuesta, facilitando al programador la implementación de la misma. De esta forma, los diagramas que conforman la documentación, podrán ser utilizados para futuras implementaciones, modificaciones o transformaciones.

2.7 Biblioteca

Colección o conjunto de subprogramas que ofrecen una funcionalidad específica al usuario. Normalmente se usa junto con otras librerías y herramientas para hacer una aplicación completa, ya que por lo general las bibliotecas no son ejecutables, pero sí pueden ser usadas por ejecutables que las necesiten para poder funcionar. (Castro, 2014)

Libqxt

Biblioteca de extensión para Qt, desarrollada bajo la licencia LGPL, que proporciona un conjunto de clases de utilidad multiplataforma para agregar funcionalidades. De las clases definidas en esta biblioteca, se utilizará QxtRPCPeer, clase que permite la transmisión de señales Qt través de una conexión de red, la señal se re-emite posteriormente al extremo receptor de la conexión. Puede funcionar además en modo de peer-to-peer(es decir, uno-a-uno, de uno a muchos o el modo cliente-servidor). En modalidad de peer o servidor, QxtRPCPeer puede escuchar y aceptar conexiones entrantes. En peer o modo cliente, se puede conectar a un servidor. Esta biblioteca permitirá comunicar componentes ubicados en distintos puntos de la red. (libqxt,2013)

2.8 Conclusiones Parciales

La metodología seleccionada promueve el trabajo en equipo, se basa en la retroalimentación continua con el cliente y está enfocada en la aplicación de pruebas que prevean errores a medida que se implementa. El lenguaje de modelado al igual que la herramienta CASE seleccionada, permitirán visualizar, especificar, construir y documentar los diagramas y clases por el cual estará compuesto la nueva versión del Framework Basado en Componentes sobre Qt. Con el *framework* de desarrollo Qt y el lenguaje de programación C++ se obtendrá una herramienta que dará cumplimiento al objetivo general de la presente investigación.

CAPÍTULO 3: CARACTERÍSTICAS DEL SISTEMA

Introducción

Durante el desarrollo de un software una buena planificación constituye la respuesta a problemas que puedan evidenciarse en todo este proceso, como el desconocimiento del tiempo estimado para el desarrollo del mismo y el esfuerzo a emplear. Esta planificación brinda al equipo de desarrollo una guía como factor clave para el éxito. La metodología XP, seleccionada para guiar el proceso de desarrollo de la nueva versión del Framework Basado en Componentes existente, se comporta de una manera diferente en cuanto al diseño de un sistema, y aunque no hace uso de técnicas como el Lenguaje de Modelado (UML), arquitectura y patrones es una metodología muy flexible y adaptable. Esta sugiere que hay que conseguir diseños simples y sencillos, procurando hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible.

Se considera que para la construcción de la propuesta de solución de acuerdo a la metodología seleccionada se deben estudiar las fases de exploración, planificación e iteraciones. Se definen las características del sistema propuesto, obteniéndose las Historias de Usuario, las iteraciones a las cuales serán asignadas las historias de usuario, el cronograma de liberación del producto y los requisitos no funcionales que debe poseer el *framework*. Una vez culminada la fase de exploración se traza la línea base de la arquitectura y se definen los patrones para el diseño de cada clase. Se confeccionan las tarjetas CRC en función de conocer el comportamiento de las clases que intervienen en la implementación y las colaboraciones que presentan entre ellas.

3.1 Fase de Exploración

En esta fase, los clientes definen a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. Debe quedar claro que las estimaciones realizadas en esta fase son primarias, puesto que estarán basadas en datos de muy alto nivel y podrían variar cuando se analicen más en detalle en cada iteración. Esta fase dura típicamente un par de semanas y el resultado es una visión general del sistema, y un plazo total estimado. (Beck, 1999)

Historias de Usuario (HU)

Una historia de usuario es una representación de un requisito de software escrito en una o dos frases utilizando el lenguaje común del usuario. Dentro de la metodología XP las historias de usuario deben ser escritas por los clientes. Son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes. Estas definen lo que se debe construir en el proyecto de software, tienen una prioridad asociada, definida por el cliente con el propósito de indicar cuáles son las más importantes para el resultado final, serán divididas en tareas y su tiempo será estimado por los desarrolladores. Generalmente se espera que la estimación de tiempo de cada historia de usuario se sitúe entre unas 10 horas y un par de semanas. Estimaciones mayores a dos semanas indican que la historia es muy compleja y debe ser dividida en varias historias. Entre sus principales características resaltan, que deben ser independientes una de otra y ser entendidas por el cliente. (Cohn, 2004)

A continuación se muestra como el equipo de desarrollo establecerá el tipo de prioridad y el tipo de riesgo para las historias de usuario. Además de representar otros artefactos que define la metodología para las etapas de Exploración, Planificación e Iteración como: la lista de reserva del producto, plan de entrega, plan de iteraciones, entre otros.

Prioridad:

- **Alta:** se le otorga a las HU que resultan funcionalidades fundamentales en el desarrollo del sistema, a las que el cliente define como principales para el control integral del sistema.
- **Media:** se le otorga a las HU que resultan para el cliente como funcionalidades a tener en cuenta, sin que estas tengan una afectación sobre el sistema que se esté desarrollando.
- **Baja:** se le otorga a las HU que constituyen funcionalidades que sirven de ayuda al control de elementos asociados al equipo de desarrollo, a la estructura y no tienen nada que ver con el sistema en desarrollo.

Riesgo en su desarrollo

- **Alto:** cuando en la implementación de las HU se considera la posible existencia de errores que lleven a la inoperatividad del código.
- **Medio:** cuando pueden aparecer errores en la implementación de las HU que puedan retrasar la entrega de la versión.

- **Bajo:** cuando pueden aparecer errores que serán tratados con relativa facilidad sin que traigan prejuicios para el desarrollo del proyecto. El cliente y los desarrolladores trabajan en conjunto para definir como agrupar las HU para su lanzamiento.

3.4 Fase de Planificación

Esta es una fase muy corta en la que el cliente establece la prioridad de cada historia de usuario con el propósito de que el programador conozca el orden en que serán implementadas. Una vez conocido el orden en que serán implementadas las HU el equipo de desarrollo estima el esfuerzo total para el desarrollo de cada una y elabora el plan de entrega para cada iteración.

3.5 Fase de Iteración

Es la fase principal en el ciclo de desarrollo de XP, donde ya una vez identificadas y descritas por el cliente, las historias de usuario y con ello la estimación del esfuerzo de cada una de ellas por el equipo de desarrollo se procede a realizar la planificación de las etapas de implementación del sistema. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

3.6 Diseño del sistema

Descripción del sistema propuesto

El *framework* cuenta con dos partes fundamentales, la primera es la encargada de la creación de componentes y la segunda de crear un entorno de trabajo para la gestión de los mismos. Dichas partes están relacionadas en función de visualizar las diferentes acciones que realizan los componentes. Permitirá realizar llamadas a funciones localizadas en otras estaciones de trabajo o en la misma máquina, que posibilitarán la comunicación de componentes para la ejecución de una acción determinada. Desde el punto de vista de un programador la llamada a la función remota (slots remotos) es y funciona de la misma manera que si fuese una llamada local. Las estaciones de trabajo tendrán la posibilidad de comportarse como cliente y servidor.

Cada componente debe contar con un archivo XML donde se especifique su nombre, su ubicación dentro del área de trabajo, la dirección ip y puerto por el que podrá conectarse a componentes localizados en otra(s) PCs y la dirección ip y puerto, que permitirán que otros ordenadores se conecten a la PC. Además, en caso de que reciba alguna señal emitida por otro componente se especifica el nombre del emisor, la señal que emite, el nombre del receptor y el evento que realiza.

Contará también el *framework* con un archivo XML donde se especificará el nombre de los componentes que se desean cargar. Además, deben contar con áreas definidas para la ubicación de componentes que posean una interfaz gráfica de usuario. En la raíz de dichas aplicaciones debe existir dos carpetas que permitan la copia del componente que se desee cargar y su archivo XML respectivamente.

Tabla 1: Personal relacionado con el framework

Personal relacionado con el framework	
Programador o Desarrollador	Es la persona encargada de realizar la implementación de nuevas funcionalidades a la versión del Framework Basado en Componentes sobre Qt ya existente en el centro. Además de comunicarse con el cliente y elaborar pruebas unitarias al sistema.

Diagrama de clases del diseño

Aunque la metodología XP no requiere de la representación de diagramas de clases, se elaboró el Diagrama de Clases del Diseño asociado a la nueva versión del Framework Basado en Componentes sobre Qt con el propósito de lograr un nivel más detallado de la solución final.

El Diagrama de Clases de Diseño representa un nivel de detalle alto, pues se relaciona con el lenguaje de programación del cual se hará uso en la implementación de un sistema. Una clase de diseño es una abstracción de una clase o construcción similar en la implementación del sistema y tienen operaciones, parámetros, atributos y tipos que son necesarios para su implementación en el lenguaje de programación elegido. (Carvajal, 2010)

En la implementación de esta nueva versión se utilizó la biblioteca libqxt para permitir que los componentes desarrollados en el *framework* interactúen con componentes distribuidos en distintas aplicaciones haciendo uso de la red. Se redefinió la interfaz AbstractComponent, la cual define las interrelaciones entre los componentes del *framework,* adicionándole nuevas instancias que permiten utilizar las funciones de la biblioteca empleada. Las clases *Application, ComponentManage, ComponentsView, Connect* y *ConnectionFactory*, fueron modificadas a partir de las nuevas funcionalidades que debe cumplir la nueva versión del *framework*, manteniendo las funciones principales para las que fueron definidas en la primera versión de la herramienta. Se adicionaron tres nuevas clases

MediaPlayer, PlayerWindow y *KonsoleP*, las cuales posibilitan la reutilización de componentes ActiveX y Kpart.

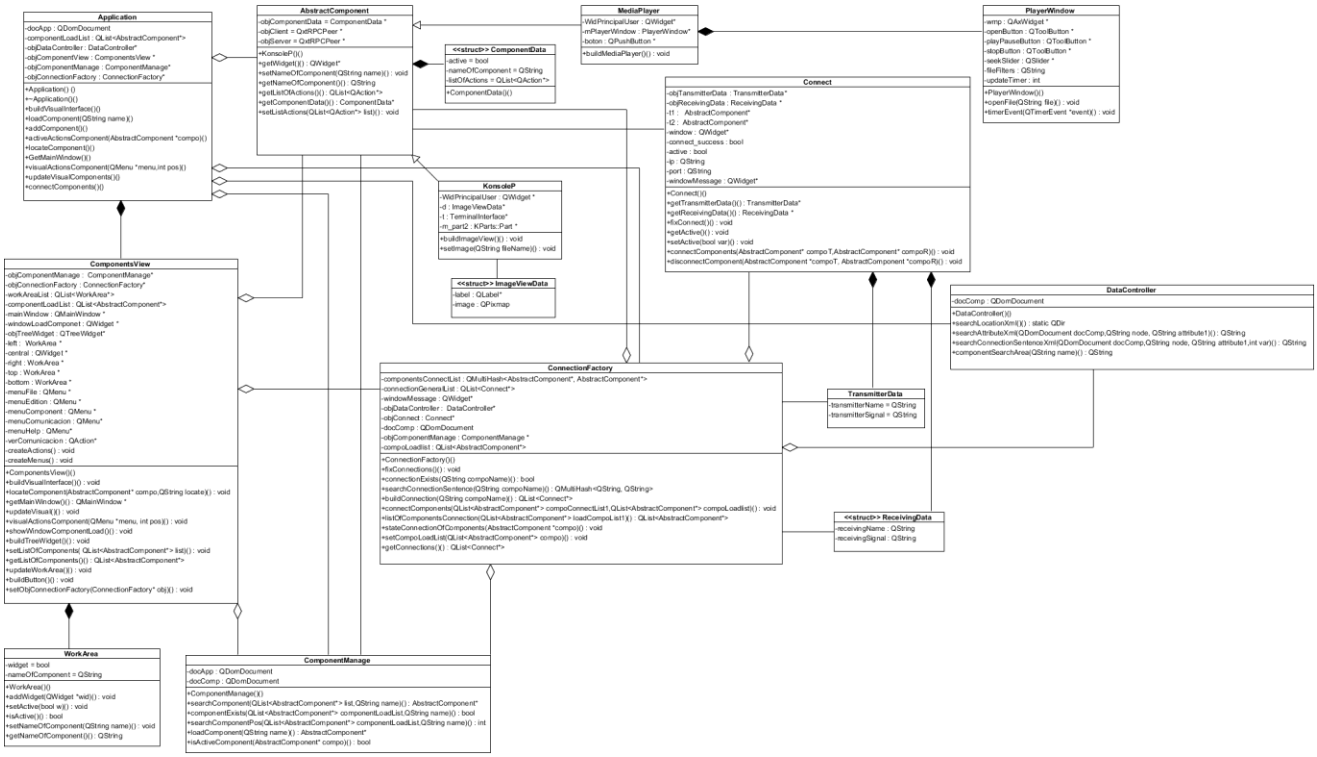

Fig. 2: Diagrama de Clases del Diseño asociado al framework basado en componentes

Arquitectura del framework

La arquitectura de software define la estructura del sistema. Esta estructura se constituye de componentes o piezas de código que nacen de la noción de abstracción, cumpliendo funciones específicas e interactuando entre sí con un comportamiento definido. Puede considerarse entonces como el "puente" entre los requisitos del sistema y la implementación. (Eckel, 2012)

En la nueva versión del Framework Basado en Componentes sobre Qt se mantiene la misma arquitectura de su primera versión, pues esta es propicia para todos los cambios que se realizarán en el mismo. Se redefinieron la gran mayoría las clases ya existentes en el *framework,* con el objetivo de permitir la interoperabilidad con componentes ubicados en distintas aplicaciones distribuidas en ordenadores a través de la red. La adición de nuevas clases posibilita la reutilización de otros entornos de componentes

como ActiveX y Kpart en el desarrollo de nuevas aplicaciones que responden a las necesidades de los proyectos del departamento Señales Digitales.

Estilos Arquitectónicos

Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales. (Reynoso, 2004)

El Estilo de Llamada y Retorno permite al diseñador de software construir una estructura de programa relativamente fácil de modificar y ajustar a escala. Se basa en la abstracción de procedimientos, funciones y métodos utilizados en grandes sistemas de software. Persigue la escalabilidad y modificabilidad.

Se aplica como subestilo del Estilo de Llamada y Retorno la arquitectura basada en componentes, pues es necesario establecer una estructura que represente al sistema a desarrollar desde todas sus perspectivas. Específicamente se utiliza esta, ya que se pretende complementar un sistema que permite la creación de componentes con una estructura definida, así como un entorno de trabajo que posibilite que dichos componentes sean cargados e interactúen entre sí.

Patrones de Diseño

Los patrones de diseño expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software. Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Para el desarrollo de la presente investigación se hará uso de algunos de los patrones GRASP y GOF existentes.

Patrones GRASP[12]

Describen los principios fundamentales de la asignación de responsabilidades a objetos expresados en forma de patrones. (Larman, 2003)

De estos patrones se utilizó el patrón Experto, pues las clases del Framework Basado en Componentes sobre Qt cuentan con la información necesaria para cumplir una responsabilidad determinada. Un ejemplo de estas es la clase *DataController*, responsable de gestionar los datos de los archivos XML correspondiente a los componentes del *framework*. Estos datos serán utilizados en la clase *ConnectionFactory* para construir las sentencias de conexión de un componente determinado.

[12] Patrones Generales de Software para Asignación de Responsabilidades

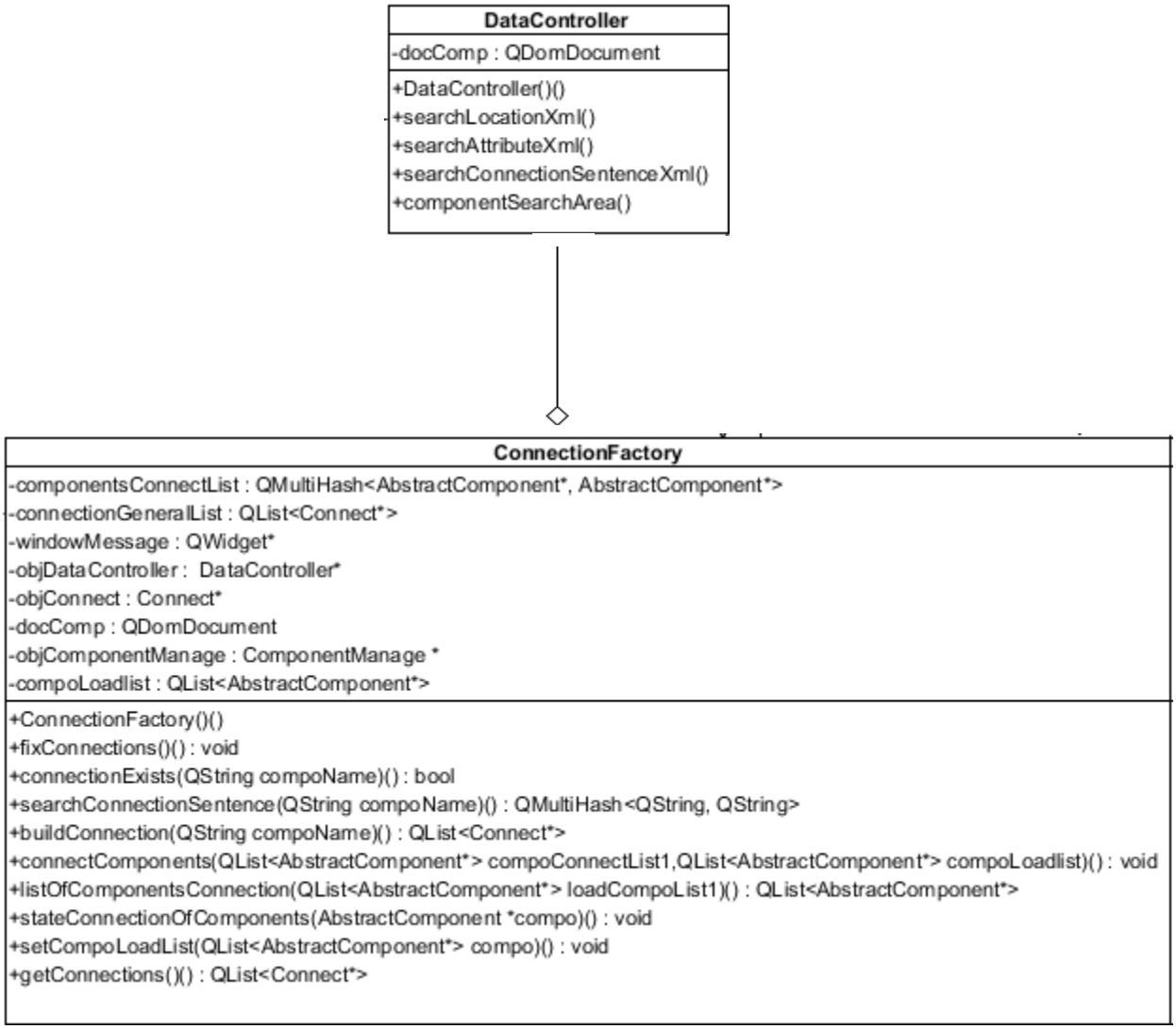

Fig. 3: Representación del patrón Experto en el Framework Basado en Componentes sobre Qt

Se empleó también el patrón Creador, el cual es el responsable de asignar a una clase la responsabilidad de crear una instancia de otra. En la clase *Application,* encargada de gestionar un área de trabajo donde los componentes se puedan cargar y comunicar entre sí, se evidencia el uso de este patrón, pues declara instancias de otras clases que utiliza estrechamente, como *DataController, ComponentsView, ComponentManage y ConnectionFactory.*

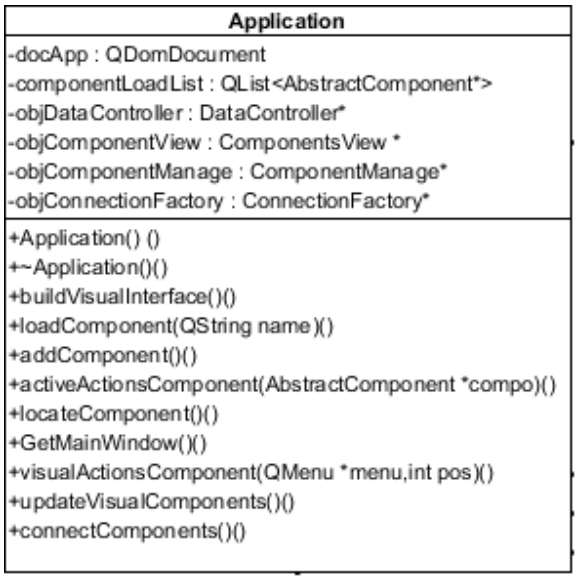

Fig. 4: Representación del patrón Creador en el Framework Basado en Componentes sobre Qt

Otro patrón utilizado es el Alta Cohesión. Consiste en asignar una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es la medida de cuan entrelazadas y enfocadas están las responsabilidades de una clase. Las clases del *framework* se caracterizan por tener responsabilidades estrechamente relacionadas y colaboran con otras clases para llevar a cabo tareas inherentes a su funcionamiento, por lo que se evidencia en todas ellas el uso de este patrón.

El patrón Bajo Acoplamiento, tiene como propósito evitar la dependencia excesiva entre las clases de un sistema, garantizando que un cambio en una clase no provoque grandes cambios en otras. Aunque en el Framework Basado en Componentes sobre Qt la gran mayoría de las clases dependen de la interfaz *AbstractComponent*, justificada esta dependencia en que esta interfaz es la que contiene toda la información referente a un componente, no se deja de mantener un acoplamiento bajo entre estas.

Otro patrón que se aplica es el Controlador ya que en el *framework* la clase *Application* posee operaciones centralizadas que manejan las acciones que se generan en el mismo, como la gestión de áreas de trabajo donde los componentes se puedan cargar, funcionar de forma autónoma y comunicarse entre sí o con otros componentes distribuidos en la red.

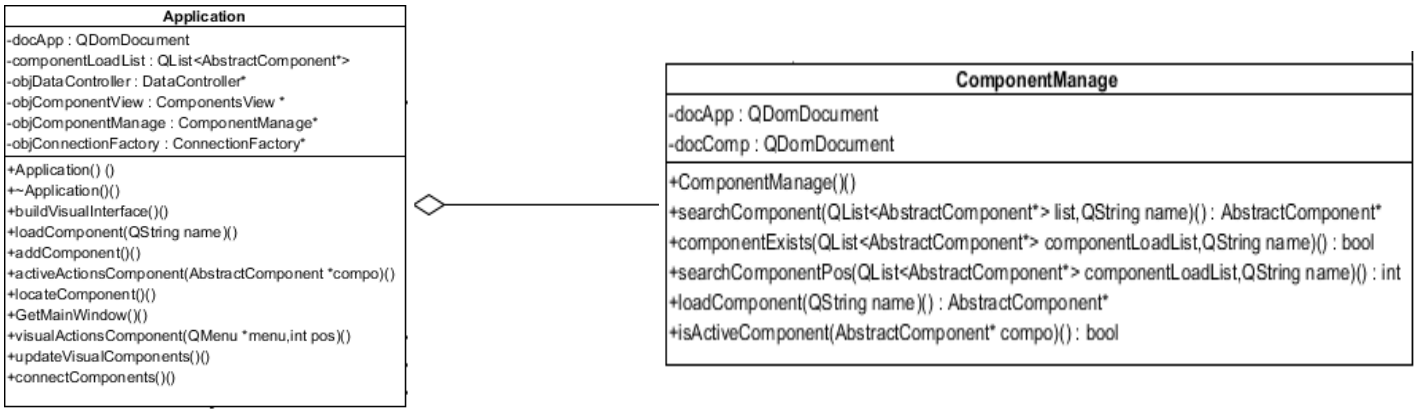

Fig. 5: Representación del patrón Controlador en el Framework Basado en Componentes sobre Qt

El *framework* Qt implementa otros patrones que fueron utilizados en el desarrollo de la solución como es el caso del patrón MetaObject, pues en la clase *ConnectionFactory* se crea un puntero a una instancia de la clase *QMetaObject*, con el objetivo de obtener información de las propiedades y métodos de un *QObject* en tiempo de ejecución.

```
AbstractComponent* component = objComponentManage->searchComponent(this->compoLoadlist, compoName);
component->listen(ip, port.toInt());
const QMetaObject* metaObject = component->metaObject();

for(int i = metaObject->methodOffset(); i < metaObject->methodCount(); ++i)
{
    QString name =  QString::fromLatin1(metaObject->method(i).signature());
    std::cout << name.toStdString() << endl;
    files.write((name + "\n").toAscii());
    QMetaMethod method = metaObject->method(metaObject->indexOfMethod(name.toAscii()));
```

Fig. 6: Evidencia del uso del patrón MetaObject

Patrones GOF [13]

Los patrones de diseño estructurales, plantean las relaciones entre clases, las combinan y forman estructuras mayores. Tratan de conseguir que los cambios en los requisitos del sistema no ocasionen cambios en las relaciones entre los objetos. De ellos se utiliza el *Adapater* (Adaptador), pues permite que clases con interfaces incompatibles puedan relacionarse mediante una clase u objeto adaptador. En el *framework* la clase *MediaPlayer* es la que cumple esta función, ya que implementa la interface *AbstractComponent* y también los métodos para comunicarse con la clase *PlayerWindow*, clase que contiene la información referente a un componente ActiveX, permitiendo que el *framework* pueda trabajar además con este tipo de componente.

[13] *Gang of Four* (Banda de cuatro)

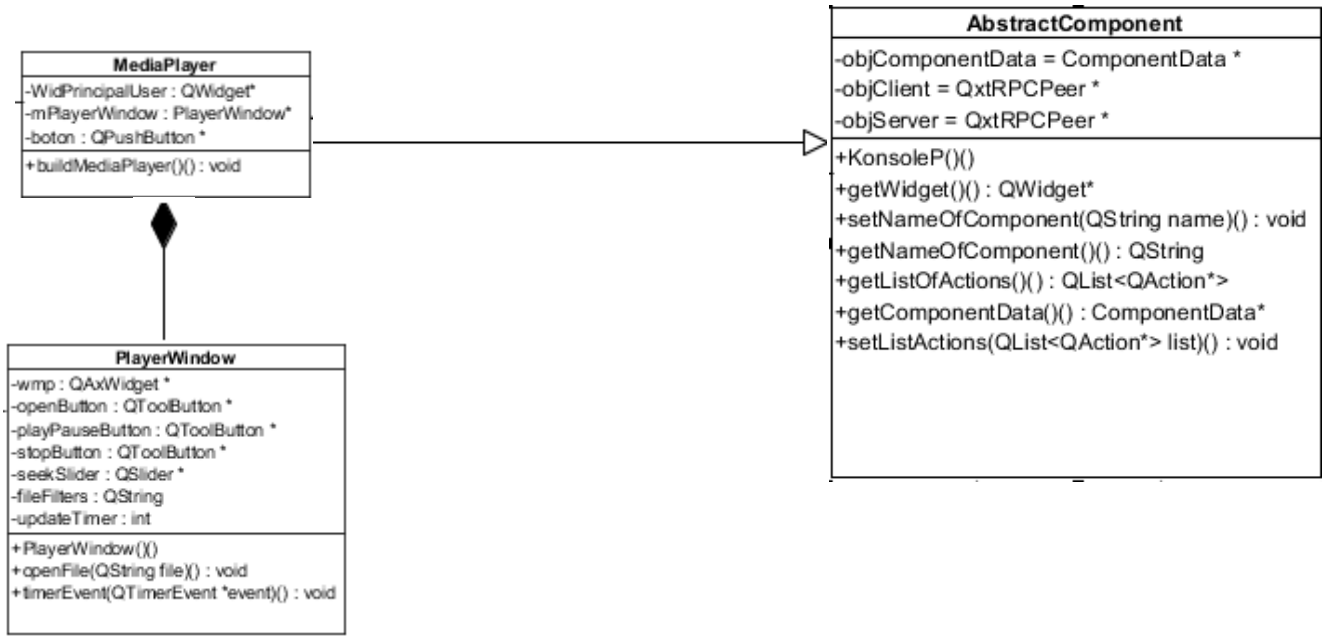

Fig. 7: Representación del patrón *Adapter* en el Framework Basado en Componentes sobre Qt

3.7 Artefactos generados

A continuación se muestran las 3 historias de usuario que fueron redactadas por el cliente.

Tabla 2: Historia de Usuario Comunicar componentes a través de la red

Historia de Usuario	
Número: 1	**Usuario:** Programador
Nombre historia: Comunicar componentes de diferentes aplicaciones a través de la red.	
Puntos estimados: 2.5	**Puntos reales:** 3
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Programador responsable: Eleanet Bujan Ponte – Ismaray Socarrás Ramírez	
Descripción: los componentes de distintas aplicaciones distribuidas por la red podrán comunicarse, intercambiando información necesaria para ejecutar una acción determinada.	
Observaciones:	

Tabla 3: Historia de Usuario Cargar componente ActiveX

Historia de Usuario	
Número: 2	**Usuario:** Programador
Nombre historia: Cargar componente ActiveX	
Puntos estimados: 1.5	**Puntos reales:** 2
Prioridad en negocio: Media	**Riesgo en desarrollo:** Alto
Programador responsable: Eleanet Bujan Ponte – Ismaray Socarrás Ramírez	
Descripción: el framework permitirá cargar los componentes ActiveX que se especifican en el archivo XML de la aplicación.	
Observaciones:	

Tabla 4: Historia de Usuario Cargar componente Kpart

Historia de Usuario	
Número: 3	**Usuario:** Programador
Nombre historia: Cargar componente Kpart	
Puntos estimados: 1.5	**Puntos reales:** 2
Prioridad en negocio: Media	**Riesgo en desarrollo:** Alto
Programador responsable: Eleanet Bujan Ponte – Ismaray Socarrás Ramírez	
Descripción: el framework permitirá cargar los componentes Kpart que se especifican en el archivo XML de la aplicación.	
Observaciones:	

Lista de reserva del producto

La lista de reserva del producto refleja los requisitos funcionales que debe cumplir el *framework* basado en componentes a desarrollar, ordenados según la prioridad en el negocio (Alta, Media). Se realiza una

estimación por semanas de cada uno, y se especifica el rol que lo estimó. La tabla contiene además los requisitos no funcionales, que requiere el sistema, teniendo en cuenta que los mismos tienen prioridad baja en el negocio.

Tabla 5: Lista de reserva del producto

Iteración	Prioridad	Descripción	Estimación	Estimado por
1	Alta	Comunicar componentes a través de la red.	3 semanas	Programador
2	Media	Cargar componente ActiveX	3 semanas	Programador y Analista
3	Media	Cargar componente Kpart	3 semanas	Programador y Analista
		Requisitos no funcionales		

Prioridad: Baja

1. Requisito de portabilidad: se requiere que el Framework Basado en Componentes sobre Qt funcione en los sistemas operativos Windows y Ubuntu 12.4 o Ubuntu 13.4.

2. Requisito de implementación: se utilizará el lenguaje de programación C++ estándar de codificación ANSI 98, el framework Qt v4.8 y el entorno de desarrollo integrado Qt Creator v2.4.

3. Requerimientos de Software: para desplegar el *framework* es necesario instalar el Framework Qt v4.8, incluyéndole la biblioteca *libqxt* y las bibliotecas de desarrollo de KDE. Podrá utilizarce en los sistemas operativos Windows y en las versiones 12.4 y 13.4 de Ubuntu.

Prioridad de las Historias de Usuario

La acción del cliente al establecer la prioridad para cada historia usuario es la que le permite conocer al programador en qué orden deberá implementar las mismas. Aclarar que las historias de usuario de prioridad alta son las más importantes y por tanto deben desarrollarse de primeras.

Tabla 6: Prioridad de las Historias de Usuario

Historia de usuario	Prioridad
Comunicar componentes a través de la red de forma remota.	Alta
Cargar componente ActiveX	Media
Cargar componente Kpart	Media

Estimación de esfuerzo de las Historias de Usuario

Según la prioridad asignada a las historias de usuario es imprescindible estimar el esfuerzo necesario para su implementación. La estimación se basa en los conocimientos y velocidad que presente el equipo de desarrollo para llevar a cabo las tareas ingenieriles o de programación. Se debe tener en cuenta que un punto equivale a una semana ideal donde los miembros del equipo de desarrollo trabajan el tiempo planeado sin interrupción. Generalmente a las historias de usuario se le asignan de 1 a 3 puntos estimados, los cuales se miden horas/hombre, días/hombre, semanas/hombre, etc; teniendo siempre presente que no siempre se cumplen de acuerdo a como se planifica, ya que pueden existir casos donde varíe la estimación efectuada.

Tabla 7: Estimación del esfuerzo necesario por Historia de Usuario

Historia de Usuario	Esfuerzo necesario (Puntos estimados)
Comunicar componentes a través de la red de forma remota.	3 (semanas)
Cargar componente ActiveX	3 (semanas)

Cargar componente Kpart 3 (semanas)

Plan de entregas

En el plan de entregas ideado para la fase de implementación, se acoplan las funcionalidades referentes a un mismo tema en módulos, permitiendo un mayor entendimiento en el desarrollo de cada historia de usuario. Dicho plan está compuesto por iteraciones de no más de tres semanas. La planificación se puede realizar basándose en el tiempo o el alcance. Al planificar según el alcance del sistema se divide la suma de puntos de las historias de usuario o puntos de esfuerzo (PE) entre la velocidad de iteración del equipo (VIE), obteniendo el número de iteraciones (NI) necesarias para su implementación.

NI= PE / VIE

La velocidad de iteración del equipo (VIE) se obtiene dividiendo la cantidad de desarrolladores (CD) entre el factor de dedicación (FD) al proyecto (en el caso de la presente investigación es de 4 [100%], luego este valor es multiplicado por el tiempo de duración máximo de una iteración (DMI), en el caso de la presente investigación es de 15 días máximo.

****

VIE= (CD / FD)*DMI

CD=1, FD =100% (4), DMI =15

VIE= 0.25*15

VIE=3.75

****

PE= 45 (9 semanas) VIE=3.75

NI= 9 / 3.75

NI =2.4

****

Luego de los cálculos efectuados se concluye que se desarrollarán 2 iteraciones, siendo el objetivo de la primera, la implementación de las HU con prioridad alta. Al final de esta iteración se contará con una primera versión de prueba, la cual será mostrada al cliente con el objetivo de obtener una retroalimentación para el grupo de trabajo. La segunda iteración tendrá como finalidad la implementación del resto de las historias de usuario con prioridad media, obteniendo una nueva versión de prueba a evaluar por el cliente.

Tabla 8: Plan de entrega de las historias de usuario

Módulos	Historia de usuario	Número de Iteración
Comunicación	1-Comunicar componentes de diferentes aplicaciones a través de la red.	Primera (4ta semana de marzo)
Carga	2- Cargar componente ActiveX 3- Cargar componente Kpart	Segunda (2da semana de abril)

Tabla 9: Plan de iteraciones

Iteraciones	Historia de usuario	Tareas	Ptos estimados
1ra	1- Comunicar componentes de diferentes aplicaciones a través de la red.	1-Redefinir la estructura del componente Abstract Component	0.1
		2- Integrar librería “libqxt” al framework.	0.1
		3- Crear conexión remota	0.6
		4- Buscar componente	0.7
		5-Conectar componentes	0.9
2da	2- Cargar componente ActiveX 3- Cargar componente ActiveX	1-Leer archivo XML.	0.3
		2- Buscar componente.	0.5
		3- Verificar existencia del componente.	0.6
		4- Ubicar componente	0.2

Tareas de ingeniería o de programación por HU

Una vez definidas las HU, los programadores dividen cada una de ellas en una serie de tareas más pequeñas, las analiza en mayor detalle y realiza una estimación de su tiempo de desarrollo. Estas tareas pueden ser descritas en un lenguaje técnico que no necesariamente garantiza el entendimiento del cliente. Finalmente, el cliente define en función de sus necesidades las HU estimadas, dejando para iteraciones posteriores aquellas que sobrepasen la capacidad productiva de la iteración. A continuación se muestran

las tareas correspondientes a la HU de prioridad alta, las restantes se muestran en los anexos de la presente investigación. (Ver Anexo# 2)

Tabla 10: Tarea de ingeniería # 3 de la HU Comunicar componentes de diferentes aplicaciones a través de la red.

Tarea	
Número tarea: 3	**Número historia:** 1
Nombre tarea: Crear conexión remota	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.6 semanas
Fecha inicio: 23/02/2014	**Fecha fin:** 18/03/2014
Programador responsable: Eleanet e Ismaray	
Descripción: dado un componente el sistema busca en su archivo XML la sentencia con el ip y puerto de la PC donde está ubicado el componente al que se desea conectar, crea la conexión y devuelve el resultado enviado por el componente en cuestión.	

Tabla 11: Tarea de ingeniería # 1 de las HU Cargar Componente ActiveX y Cargar Componente Kpart

Tarea	
Número tarea: 1	**Número historia:** 2 y 3
Nombre tarea: Leer del archivo XML de la aplicación.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.3 semanas
Fecha inicio: 19/03/2014	**Fecha fin:** 22/03/2014
Programador responsable: Eleanet y Ismaray	
Descripción: el sistema obtiene del archivo XML de la aplicación el nombre de los componentes que serán cargados.	

Tarjetas CRC (Clase, Responsabilidad y Colaborador)

Se emplean con el objetivo de facilitar la comunicación en un equipo de desarrollo y documentar los resultados. Para cada una de las clases que brindan una funcionalidad directa al negocio, se elabora una tarjeta de este tipo, especificando su finalidad y las clases con las que interactúa.

Tabla 12: Tarjeta CRC #1 "Comunicación"

Tarjeta CRC	
Clase: Comunicación	
Responsabilidades:	**Colaboraciones:**
Crear conexión remota	XML
Buscar componente	
Conectar componentes	

Tabla 13: Tarjeta CRC #2 "Carga"

Tarjeta CRC	
Clase: Carga	
Responsabilidades:	**Colaboraciones:**
Buscar componente.	XML
Verificar existencia del componente.	
Ubicar componente	

Tabla 14: Tarjeta CRC #3 "XML"

Tarjeta CRC	
Clase: XML	
Responsabilidades:	**Colaboraciones:**
Leer del archivo XML	

3.8 Conclusiones Parciales

La construcción de cualquier sistema debe estar respaldada por un buen diseño arquitectónico, por lo que se concluye que la aplicación del estilo de Llamada y Retorno garantizó que el *framework* sea escalable y modificable, posibilitando que pueda ser mejorado en futuras versiones. La simplicidad que garantiza la realización de las tarjetas CRC permitió que estas puedan ser modificadas con facilidad frente a posibles cambios o actualizaciones con respecto al diseño.

CAPÍTULO 4. PRUEBAS DE SOFTWARE

Introducción

Las pruebas de software consisten en la verificación del comportamiento de un programa en un conjunto finito de casos de prueba. Son una serie de actividades que se realizan con el propósito de encontrar los posibles fallos de implementación, calidad o usabilidad de un programa probando el comportamiento del mismo. En el presente capítulo se realizarán pruebas de caja blanca al código fuente del *framework*, hacienda uso del método Camino Básico, en función de identificar y corregir fallos cometidos durante el desarrollo de las HU. Se aplicarán además pruebas de regresión para comprobar que los cambios realizados en el *framework* no introducen un comportamiento no deseado o errores adicionales en otros módulos o partes no modificados.

4.1 Pruebas Unitarias

Las pruebas unitarias son una de las piedras angulares de XP. Todos los módulos deben pasar estas pruebas antes de ser liberados o publicados. Su objetivo es comprobar que el módulo, entendido como una unidad funcional, está correctamente codificado. Se le realizan pruebas a pequeños fragmentos del código, encargados de una tarea específica y se asegura que funcionen tal y como lo define la especificación del programa. Durante la prueba de unidad, la comprobación selectiva de los caminos de ejecución es una tarea esencial. Se deben diseñar casos de prueba para detectar errores debidos a cálculos incorrectos, comparaciones incorrectas o flujos de control inapropiados. La prueba del camino básico es una técnica muy efectiva para descubrir una gran cantidad de errores en los caminos.

Pruebas de Caja Blanca

Las pruebas de caja blanca realizan, de alguna manera, un seguimiento del código fuente según va ejecutando los casos de prueba, de manera que se determinan de manera concreta las instrucciones que han sido ejecutadas por los casos de prueba. Es por ello que se considera como uno de los tipos de pruebas más importantes que se le aplican a los *software*, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. El procedimiento para llevar a cabo la técnica de prueba del camino básico es el siguiente:

Paso 1: Dibujar el grafo de flujo asociado al método *searchConnectionSentence*. (Ver Anexo3)

Para la confección del grafo de flujo asociado al método *searchConnectionSentence* es necesario primeramente conocer los conceptos de los elementos que lo integran.

Grafo: es un par (V, A) donde V es un conjunto finito de elementos que se denominan vértices y A es un conjunto de pares no ordenados <x, y>, donde x∈V, y∈V, denominados aristas o arcos.

Nodo: es una o más secuencias de procesos, que pueden ser instrucciones o sentencias de decisión, pueden haber nodos sin asociar que sólo se utilizan al principio o al final del código.

Arista: representan el flujo de información.

Región: Área limitada entre nodos y aristas.

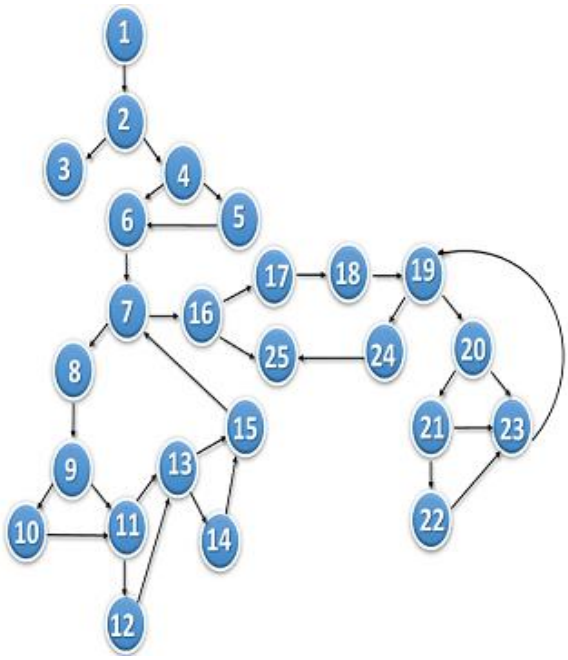

Fig. 8: Grafo de flujo asociado al método *searchConnectionSentence*

Paso 2: Cálculo de la Complejidad Ciclomática (V (G))

La complejidad ciclomática define el número de caminos independientes, que a su vez proporciona el mínimo número de pruebas que se debe realizar, permitiendo que una secuencia se ejecute al menos una vez. Para esto se plantea la siguiente fórmula:

V (G) = (A - N) + 2, teniendo que A es el número de aristas y N es el número de nodos del grafo de flujo asociado, sustituyendo V (G) = (33-25) + 2, V (G) =10.

Se concluye que existen 10 caminos independientes por los que el flujo del grafo puede circular y el mínimo número de casos de pruebas que deberán ejecutarse.

Paso 3: Determinar el conjunto básico de caminos independientes.

Un camino independiente constituye cualquier camino o vía de ejecución del código o programa que introduce un nuevo conjunto de sentencias de procesos o una nueva condición, que deberá abarcar al menos una arista que aún no haya sido recorrida.

Camino básico #1: 1, 2, 3

Camino básico #2: 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 16, 25

Camino básico #3: 1, 2, 4, 5, 6, 7, 8, 9, 11, 13, 15, 7, 16, 25

Camino básico #4: 1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 16, 25

Camino básico #5: 1, 2, 4, 6, 7, 8, 9, 11, 13, 15, 7, 16, 25

Camino básico #6: 1, 2, 4, 5, 6, 7, 16, 25

Camino básico #7: 1, 2, 4, 6, 7, 16, 17, 18, 19, 24, 25

Camino básico #8: 1, 2, 4, 6, 7, 16, 17, 18, 19, 20, 21, 22, 23, 19, 24, 25

Camino básico #9: 1, 2, 4, 5, 6, 7, 16, 17, 18, 19, 24, 25

Camino básico #10: 1, 2, 4, 5, 6, 7, 16, 17, 18, 19, 20, 21, 22, 23, 19, 24, 25

Paso 4: Casos de Pruebas

Los Casos de pruebas definen un conjunto específico de entradas de pruebas, ejecución de condiciones y resultados esperados. Luego de la elaboración de los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

A continuación se muestran dos casos de prueba asociados al método *searchConnectionSentence*, los restantes se pueden encontrar en el **Anexo4.**

Tabla 15: Caso de prueba para el camino básico No.6 de la funcionalidad *searchConnectionSentence*

Caso de prueba para el camino básico No.6
Camino: 1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 16, 25
Descripción: Verifica si el archivo XML abierto no es de solo lectura. Obtiene los datos del componente y agrega a una lista aquellos datos que no estén vacíos.
Entrada: Nombre del componente.

Resultado esperado: Lista con las señales y slots asociados a un componente
Resultado obtenido: Se obtuvo una lista con las señales y slots asociados a un componente

Tabla 16: Caso de prueba para el camino básico No.8 de la funcionalidad *searchConnectionSentence*

Caso de prueba para el camino básico No.8
Camino: 1, 2, 4, 6, 7, 16, 17, 18, 19, 20, 21, 22, 23, 19, 24, 25
Descripción: Verifica si el archivo XML abierto no es de solo lectura. Obtiene los datos del componente y agrega a una lista aquellos datos que no estén vacíos.
Entrada: Nombre del componente.
Resultado esperado: Lista con las señales, slots, ip y puerto de un componente local o remoto
Resultado obtenido: Se obtuvo una lista con las señales, slots, ip y puerto de un componente local o remoto

Grafo de flujo asociado al método *connectComponents*. (Ver Anexo3)

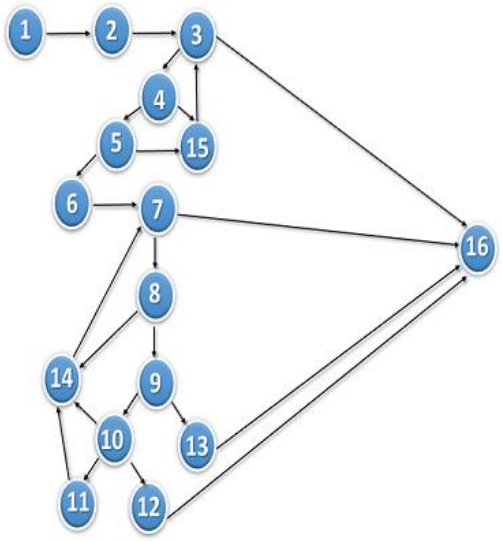

Fig. 9: Grafo de flujo asociado al método *connectComponents*

Cálculo de la Complejidad Ciclomática (V (G))

V (G) = (A - N) + 2, teniendo que A es el número de aristas y N es el número de nodos del grafo de flujo asociado, sustituyendo V (G) = (23-16) + 2, V (G) = 9.

Conjunto básico de caminos independientes:

Camino básico #1: 1, 2, 3, 16

Camino básico #2: 1, 2, 3, 4, 5, 15, 3, 16

Camino básico #3: 1, 2, 3, 15, 3, 16

Camino básico #4: 1, 2, 3, 4, 5, 6, 7, 16

Camino básico #5: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14, 7, 16

Camino básico #6: 1, 2, 3, 4, 5, 6, 7, 8, 14, 7, 16

Camino básico #7: 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 16

Camino básico #8: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 16

Camino básico #9: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 7, 16

Casos de Pruebas:

A continuación se muestran dos casos de prueba asociados al método *connectComponents*, los restantes se pueden encontrar en el **Anexo4.**

Tabla 17: Caso de prueba para el camino básico No.9 de la funcionalidad *connectComponents*

Caso de prueba para el camino básico No.9
Camino: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 7, 16
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión y se procede a la construcción de las conexiones del componente. Se verifica que exista conexión, en caso de existir se procede a recorrer la lista de conexiones con el propósito de asignar valores a los componentes emisor y receptor. Se verifica que el componente emisor haya sido cargado y en caso de que esto se cumpla se realiza la conexión.
Entrada: Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado: La comunicación entre los componentes cargados.
Resultado obtenido: Se logró la comunicación entre los componentes cargados.

Tabla 18*:* Caso de prueba para el camino básico No.2 de la funcionalidad *connectComponents*

Caso de prueba para el camino básico No.2
Camino: 1, 2, 3, 4, 5, 15, 3, 16
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión y se procede a la construcción de las conexiones del componente. En caso de no existir ninguna conexión entre componentes no se realiza la comunicación entre componentes.
Entrada: Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado: Los componentes cargados no se comunicarán.
Resultado obtenido: No se logró la comunicación entre los componentes cargados.

4.2 Pruebas funcionales

Prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Las pruebas funcionales se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático. Son pruebas específicas, concretas y exhaustivas para probar y validar que el software hace lo que debe y sobre todo, lo que se ha especificado.

Pruebas de regresión

Su objetivo general es descubrir las causas de nuevos errores, carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, inducidos por cambios recientemente realizados en partes de la aplicación que anteriormente al citado cambio no eran propensas a este tipo de error. Estas pruebas se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error como para realizar una mejora. No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre el mismo u otros componentes.

A continuación se describen los casos de prueba asociados a algunas de las funcionalidades que no sufrieron cambios durante el desarrollo de la nueva versión del *framework,* comprobando que su funcionamiento no haya sido afectado por las modificaciones realizadas. **(Ver Anexo3)**

Tabla 19: Caso de prueba asociado a la funcionalidad *loadComponent*

Caso de prueba
Número: 1
Entrada*:* Nombre de un componente.
Descripción: una vez que el framework obtiene el nombre de un componente verifica que no haya sido cargado con anterioridad. En caso de que no haya sido cargado, activa las acciones del componente y lo añade a una lista de componentes cargados.
Resultado esperado: El *framework* carga correctamente el componente.
Resultado obtenido: El componente se cargó correctamente.

Tabla 20: Caso de prueba asociado a la funcionalidad *searchComponent*

Caso de prueba
Número: 2
Entrada*:* Lista de componentes y nombre de un componente.
Descripción: dada una lista con los componentes del *framework se* obtienen los datos asociados al componente con el mismo nombre entrado por parámetro.
Resultado esperado: Se obtienen los datos de un componente específico.
Resultado obtenido: Se obtuvieron los datos de un componente específico.

Tabla 21: Caso de prueba asociado a la funcionalidad *connectionExists*

Caso de prueba
Número: 3

Entrada*:* Nombre de un componente
Descripción: dado el nombre de un componente se comprueba si este tiene conexión con otros componentes.
Resultado esperado: Se comprueba que un componente específico tiene conexión con otros componentes.
Resultado obtenido: Se comprobó que un componente dado su nombre, tiene conexión con otros componentes.

4.3 Conclusiones

Una vez implementada la propuesta de solución se procedió a aplicarle las pruebas de software, uno de los aspectos claves de la metodología XP, permitiendo verificar si el *framework* cumple con las características esperadas. Una vez recorridos los caminos independientes del Grafo de Flujo asociado a algunas de las funcionalidades del *framework* se concluye que los resultados obtenidos se corresponden con los esperados por el equipo de desarrollo. La aplicación de las pruebas de regresión permitió comprobar que los cambios realizados a la nueva versión del Framework Basado en Componentes sobre Qt, no alteraron el funcionamiento de aquellas funcionalidades que no fueron modificadas.

CONCLUSIONES GENERALES

- El análisis de soluciones existentes, con características y funcionamiento similar al framework desarrollado, aportaron ideas para la concepción del mismo.
- Las tecnologías y herramienta seleccionadas, garantizaron el desarrollo del sistema en todas sus etapas, obteniendo un software con la calidad requerida.
- La realización de un buena exploración y planificación, fases de la metodología seleccionada, dieron paso a una implementación efectiva para los desarrolladores del sistema.
- La arquitectura de software y los patrones de diseño permitieron establecer una correcta estructura del *framework*, posibilitando al mismo tiempo una codificación organizada y entendible.
- La arquitectura orientada a componentes permite diseñar la nueva versión del módulo de manera flexible, para incluir nuevas operaciones que no se encuentran en la primera versión y pudieran ser necesarias para un entorno de trabajo determinado.
- Las pruebas realizadas permitieron comprobar los errores existentes en el código, para ser resueltos y por ende garantizar una mayor calidad y confiabilidad en el *framework*.

RECOMENDACIONES

- Extender la solución al centro GEYSED con el objetivo de que sea utilizado por los proyectos que lo integran.
- Permitir la interacción con componentes a través de la web; posibilitando que los mismos puedan ser manejados tanto por aplicaciones de web como de escritorio.

BIBLIOGRAFÍA REFERENCIADA

- **Ayuda de GCF 2.6.** 2012. VCreate Logic . VCreate Logic . [En línea] 2012. [Citado el: 5 de diciembre de 2013.] http://gcf.vcreatelogic.com.
- **Beck, K.** 1999. "Extreme Programming Explained. Embrace Change". s.l. : Pearson Education, 1999.
- **Borges, Adrián Martínez y Yunet Gasca Suárez.** 2013. . Framework para el desarrollo basado en componentes sobre Qt . Habana : s.n., 2013. .
- **Carvajal, Erlys.** 2010. Análisis y diseño del subsistema de Análisis de Resultados de un Simulador de Procesos Químicos . La Habana : s.n., 2010.
- **Casal Terreros, Julio** . 2014. Desarrollo de software basado en componentes. 2014.
- **Castro, Luis.** 2014 . about. [En línea] 2014 . [Citado el: 8 de diciembre de 2013.] http://aprenderinternet.about.com/od/Glosario/fl/Que-es-una-libreria.htm.
- **Cohn, Mike.** 2004. "User Stories Applied". s.l. : Addison Wesley, 2004. ISBN 0-321-20568-5.
- 2014. **Corba.** Corba. [En línea] Object Management Group, 2014. [Citado el: 7 de diciembre de 2013.] http://www.corba.org/.
- **Dueñas Gómez, Laureano Felipe.** 2007. Interoperabilidad en los Sistemas de Información Documental (SID): la información debe fluir . 2007.
- **Eckel, Bruce.** 2012. Pensar en C++ (Volumen 2). 2012.
- **Gonzalo Génova Fuster, José Miguel Fuentes Torres, María Cruz Valiente Blázquez.** 2013. Evaluación comparativa de herramientas CASE para UML desde el punto de vista notacional. . Madrid : s.n., 2013.

BIBLIOGRAFÍA REFERENCIADA

- **Guérin, Brice Arnaud.** 2011. Programación Unix & Windows Standar Template Library.Creación de un programa archivado C++. 2011.
- 2012. **IBM** . IBM . [En línea] 2012. [Citado el: 5 de diciembre de 2013.] http://www.ibm.com/us/en/.
- **James Rumbaugh, IvarJacobson, Grady Booch.** 1998. La referencia definitiva de UML escritas por sus creadores. . s.l. : Addison Wesley, 1998.
- **Krutchen, Philippe.** 2000. Rational Unified Process. 2000.
- **Larman, Craig.** 2003. UML y PATRONES. Introducción al análisis y diseño orientado a objetos. 2003.
- 2013. **libqxt**. [En línea] Atlassian, 15 de 07 de 2013. [Citado el: 10 de enero de 2014.] http://dev.libqxt.org/libqxt/wiki/Home.
- **Marcos, E.** 2005. "Investigación en Ingeniería del Software vs. Desarrollo Software" . Universidad Rey Juan Carlos. : Grupo KYBELE, 2005.
- **Martínez, León Alberto.** 2011. Implementación de un editor gráfico de circuitos eléctricos con Qt . Cartagena : s.n., 2011.
- **McClure, Carma.** 1989.. The CASE Experience. s.l. : BYTE, 1989.
- 2008. **Microsoft** . Microsoft. [En línea] 18 de enero de 2008. [Citado el: 3 de diciembre de 2013.] http://msdn.microsoft.com/enus/library/aa751972%28VS.85%29.aspx.
- **Montilva C., Juan A, Arapé, Nelson and Colmenares, Juan Andrés.** 2003. Desarrollo de Software basado en Componentes. Mérida : s.n., 2003.
- **Palomino González, Antonio.** 2009. Definición de interfaz. 2009.
- **Reynoso, Carlos Billy.** 2004. Introducción a la Arquitectura de Software. Habana : s.n., 2004.
- **Riehle, Dirk.** 2000 . Framework Design: A Role Modeling Approach. Zurich : s.n., 2000 .

BIBLIOGRAFÍA REFERENCIADA

- **Sametinger.** 1997. Software engineering withreusable components. s.l. : Springer Verlag, 1997.
- **Sommerville, Ian.** 2000. Software engineering 6ta edición . s.l. : Addison-Wesley , 2000.
- **Szyperski, Clemens.** 2002. Component Software: Beyond Object Oriented Programming. 2002.
- **Viñolo, Raydel Raúl and Roquero Figueroa, Alexander.** 2012. Sistema Gestor de Procesos de Media v2. 2012.

BIBLIOGRAFÍA CONSULTADA

Agüero, Pedro Manuel Zayas. 2010. eumed. [En línea] 2010. [Citado el: 3 de noviembre de 2013.] http://www.eumed.net/libros-gratis/2010e/822/Metodos%20del%20conocimiento%teorico.htm.

André Ampuero, M., V. D. Muñoz Castillo. "¿Metodologías tradicionales o metodologías ágiles?", 2008.

B.Terry & D.Logee. 1990. *Terminology for Software Engineering and Computer-aided Software Engineering.* 1990.

Booch, G., Rumbaugh, J., Jacobson, I, 1999.El Lenguaje Unificado de Modelado, Addison Wesley.

Bjarne Stroustrup, The C++ Programming Language, Addison-Wesley Pub Co; Tercera edición (15 de febrero de 2000).

Calero, Wynnie. 2010. ingenieraupoliana.blogspot. [En línea] 8 de octubre de 2010. [Citado el: 5 de noviembre de 2013.] http://ingenieraupoliana.blogspot.com/2010/10/modelo-de-desarrollo-basado-en.html.

Casal Terreros, Julio (2013). Desarrollo de Software Basado en Componentes

Coad, P., Jeff De Luca, E. Lefebvre. "Java Modeling In Color With UML: Enterprise Components and Proces"s. Prentice Hall, 1999.

Cockburn, Alistair. Agile Software (2008), Development. Highsmith Series.

Cockbun, A. "Agile Software Development". Addison-Wesley, 2001.

Chin, Gary (2004). Agile Project Management: How to Succeed in the Face of Changing Project Requirements.

García, Francisco José and Laguna, Miguel Ángel. Construcción de framework basada en análisis de conceptos formales y soportada por Mecanos. Valladolid : s.n.

Hernández, Dr. Francisco Bijarro. scribd. [En línea] [Citado el: 6 de noviembre de 2013.] http://es.scribd.com/doc/15952439/53/Historico-Logico.

BIBLIOGRAFÍA CONSULTADA

Highsmith, J.A. "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems". Dorset House, 2000.

James Rumbaugh, Ivar Jacobson, Grady Booch. La referencia definitiva de UML escritas por sus creadores. s.l. : Addison Wesley.

Joskowicz, José;. 2008.Reglas y Prácticas en Extreme Programming. 2008.

Letelier, Patricio and Penadés, Carmen. 2006.Métodologías ágiles para el desarrollo de software: Extreme Programming (XP).

Lucy Nohemy Medina Velandia, Álvaro Escobar, Andrés Abel Arenas Prada. 2009. LACCEI Latin American and Caribbean Conference for Engineering and Technology. [En línea] 2-5 de Junio de 2009. [Citado el: 25 de noviembre de 2013.] http://www.laccei.org/LACCEI2009-Venezuela/p173.pdf.

Martin Fowler, Kendall Sccott, "UML Gota a Gota", 1999.

Martínez, Gustavo (2011). Coding, quality check and documentation (300%): Get them from the same development team.

McClure, Carma. 1989. *The CASE Experience.* s.l: BYTE, 1989. 235

Montes de Oca, Altair. scribd. [En línea] [Citado el: 7 de noviembre de 2013.] http://es.scribd.com/doc/47889658/MODELO-BASADO-EN-COMPONENTES.

Montilva C., Juan A, Arapé, Nelson and Colmenares, Juan Andrés. 2003. Desarrollo de Software basado en Componentes. Mérida : s.n., 2003.

Perdita Stevens, Utilización de UML en Ingeniería del Software con Objetos y Componentes.

Pérez. Ramírez D. "Investigación de la metodología ágil Extreme Programming y su aplicación a un caso de estudio". Y Guntín Oliveros Y., Coello Mena J. (Tutor). Tesis de Grado. La Habana, Instituto Superior Politécnico José Antonio Echeverría. 2008.

BIBLIOGRAFÍA CONSULTADA

Pressman, Roger S. Ingeniería de Software, un enfoque práctico. Quinta edición. S.l.: McGraw-Hill Companies, 2002. ISBN: 8448132149.

Reynoso, C. "Métodos Heterodoxos en Desarrollo de Software".2004.

Sayas, Carlos Alvarez de. 1995. Metodología de la investigación científica. Santiago de Cuba, Cuba : s.n., 1995. Car95.

Schwaber, K. y M. Beedle. "Agile Software Development with SCRUM".
2001.

Sierra Bravo. Técnicas de investigación social. 8va. Edición. Editorial Paraninfo. 1996. 193p.

Stapleton, J. "Dsdm Dynamic Systems Development Method: The Method in Practic"e. Addison-Wesley, 1997.

Sommerville. 1999. *Edición 8va Cap 19 Ingeniería de Software Basada en Componentes.* 1999. Som99**.**

Sutherland, J., A. Victorov, y J. Blount. "Adaptive Engineering of Large Software Projects with Distributed/Outsourced Teams". 2006.

Szyperski. 1998. Component Software-Beyond Objetc-Oriented Programming. 1998.

Szyperski, Clemens. 2002. Component Software: Beyond Object Oriented Programming. 2002.

Usaola, Dtor. Macario Polo. 2006. Curso de doctorado sobre software y gestión del conocimiento. Ciudad Real : s.n., 2006. Dto06.

Viñolo, Raydel Raúl and Roquero Figueroa, Alexander. 2012**.**Sistema Gestor de Procesos de Media v2. 2012.

Yislen Dolores Ramírez Camejo, Heydi Menéndez Avalos. 2009. *Proceso de Pruebas de Liberación al Sistema de Manejo de Datos de Ensayos Clínicos Cubano.* Habana : s.n., 2009. Yis09.

Winnick Cluts, Nancy. Microsoft, Creating ActiveX Components in C++. [En línea] [Citado el: 3 de diceimbre de 2013.] http://msdn.microsoft.com/en-us/library/ms974283.aspx

ANEXOS

Anexo#1 Entrevistas

Fecha: 27 de febrero del 2014

Nombre del entrevistado:

Reynier Pupo Gómez

1. ¿Cuál considera usted que será el aporte de la investigación en cuestión?

Con esta segunda versión al Framework Basado en Componentes sobre Qt sería posible comunicar componentes de forma remota a través de la red y cargar componentes ampliamente difundidos como ActiveX y Kpart. Esto le servirá de ayuda a los proyectos del departamento de Señales Digitales, como por ejemplo al proyecto Video Vigilancia porque podrían cargarse controles específicos que vienen con las cámaras para manipularlas en sus aplicaciones. La comunicación entre componentes remotos posibilitaría la comunicación entre componentes de aplicaciones en distintas máquinas y respondería a las características de la arquitectura distribuida que poseen la mayoría de los proyectos del departamento.

Anexo # 2. Tareas Ingenieriles

Tabla 22: Tarea de ingeniería # 2 de la HU Cargar componente Kpart y ActiveX

Tarea	
Número tarea: 2	**Número historia:** 2 y 3
Nombre tarea: Buscar componente**.**	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.5 semanas
Fecha inicio: /03/2014	**Fecha fin:** 27/04/2014
Programador responsable: Ismaray y Eleanet	
Descripción: el sistema busca el componente especificado en el archivo XML de la aplicación y en caso de existir crea una instancia del mismo.	

Tabla 23: Tarea de ingeniería # 3 de la HU Cargar componente Kpart y ActiveX

Tarea	
Número tarea: 3	**Número historia:** 2 y 3
Nombre tarea: Verificar existencia del componente.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.6 semanas
Fecha inicio: 28/03/2014	**Fecha fin:** 30/03/2014
Programador responsable: Ismaray y Eleanet	
Descripción: el sistema verifica si el componente ha sido cargado por la aplicación con anterioridad. En caso de que ya haya sido cargado anteriormente el sistema notifica que un componente solo puede ser cargado por la aplicación una sola vez y en caso contrario procede a cargarlo.	

Tabla 24: Tarea de ingeniería # 4 de la HU Cargar componente Kpart y ActiveX

Tarea	
Número tarea: 4	**Número historia:** 3
Nombre tarea: Ubicar componente	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.2 semanas
Fecha inicio: 1/04/2014	**Fecha fin:** 3/04/2014
Programador responsable: Eleanet e Ismaray	
Descripción: destacar que solo son ubicados aquellos componentes que posean una interfaz gráfica de usuario. El sistema lee del archivo XML del componente con el propósito de indicar en qué área de la aplicación debe ser ubicado el componente. En caso de que el área especificada esté ocupada por otro componente en la aplicación es creada una nueva área y ubicado en la misma el componente.	

Tabla 25: Tarea de ingeniería # 1 de la HU Comunicar componentes de diferentes aplicaciones a través de la red.

Tarea	
Número tarea: 1	**Número historia:** 1
Nombre tarea: Redefinir la estructura del componente Abstract Component	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.1 semanas
Fecha inicio: 1/03/2014	**Fecha fin:** 2/03/2014
Programador responsable: Eleanet e Ismaray	
Descripción: se le agrega a la estructura del Abstract Component, un método que le permita escuchar las señales (conexiones) que son enviadas desde otro ordenador a través de la red. Se crean dos nuevos objetos, instancia de la librería libqxt, con la función de comportarse como cliente y servidor	

Tabla 26: Tarea de ingeniería # 2 de la HU Comunicar componentes de diferentes aplicaciones a través de la red.

Tarea	
Número tarea: 2	**Número historia:** 1
Nombre tarea: Integrar librería “libqxt” al framework	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.1 semanas
Fecha inicio: 3/03/2014	**Fecha fin:** 4/03/2014
Programador responsable: Eleanet e Ismaray	
Descripción: se integra la librería libqxt al framework, pues es la biblioteca que permitirá establecer conexión con componentes ubicados en distintas estaciones de trabajo.	

Tabla 27: Tarea de ingeniería # 4 de la HU Comunicar componentes de diferentes aplicaciones a través de la red.

Tarea	
Número tarea: 4	**Número historia:** 1
Nombre tarea: Buscar componente	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.7 semanas
Fecha inicio: 5/03/2014	**Fecha fin:** 7/03/2014
Programador responsable: Eleanet e Ismaray	
Descripción: el sistema busca el componente al que desea acceder una aplicación ubicada en otra PC, el cual está especificado en su archivo XML.	

Tabla 28: Tarea de ingeniería # 5 de la HU Comunicar componentes de diferentes aplicaciones a través de la red.

Tarea	
Número tarea: 4	**Número historia:** 1
Nombre tarea: Conectar componente	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.9 semanas
Fecha inicio: 7/03/2014	**Fecha fin:** 28/03/2014
Programador responsable: Eleanet e Ismaray	
Descripción: el sistema busca el componente al que desea acceder una aplicación ubicada en otra PC, el cual está especificado en su archivo XML, del mismo lee el ip y puerto al que se desea conectar.	

.

Anexo# 3: Numeración de los métodos escogidos para realizar el grafo de flujo.

```
QMultiHash<QString, QString> ConnectionFactory::searchConnectionSentence(QString compoName)
{
    QMultiHash<QString,QString> sentenceList;
    QString transmitter;
    QString receiving;
    QString ip="-1";
    QString port="-1";                                                          } 1
    QDir direction=objDataController->searchLocationXml();
    QFile file(direction.absolutePath()+"/"+compoName+".xml");

   if (!file.open(QIODevice::ReadOnly)) ---------------------------------------- 2
       return sentenceList;     ----------------------------------------------- 3
    if (!docComp.setContent(&file))-------------------------------------------- 4
       file.close();  --------------------------------------------------------- 5
         6                                   7                                          15
    for(int i=0;i<docComp.childNodes().at(0).toElement().elementsByTagName("Componente").size();i++) {
    transmitter.clear();
    receiving.clear();   } 8
     if(docComp.childNodes().at(0).toElement().elementsByTagName("Componente").at(i).toElement().attributeNode("emite").name()=="emite"){------------9
        transmitter=objDataController->searchConnectionSentenceXml(docComp,"Componente","emite",i);
        receiving=objDataController->searchConnectionSentenceXml(docComp,"Componente","recibe",i);  } 10
        }

     if(docComp.childNodes().at(0).toElement().elementsByTagName("Componente").at(i).toElement().attributeNode("ip").name()=="ip"){-----------------11
        ip = docComp.childNodes().at(0).toElement().elementsByTagName("Componente").at(i).toElement().attributeNode("ip").name();
        port = docComp.childNodes().at(0).toElement().elementsByTagName("Componente").at(i).toElement().attributeNode("port").name();  } 12

        }

     if(transmitter!="" && receiving!="") --------------------------------------- 13
          sentenceList.insert(transmitter,receiving);  ------------------------- 14
    }

       if (ip != "-1") ---------------------------------------------------------- 16
       {
           AbstractComponent* component = objComponentManage->searchComponent(this->compoLoadlist, compoName);
           component->listen(ip, port.toInt());

                                                                                 } 17
           const QMetaObject* metaObject = component->metaObject();
           QFile files("prueba.txt");
           files.open(QIODevice::WriteOnly);
           files.write("prueba\n");
                  18                              19                  23
           for(int i = metaObject->methodOffset(); i < metaObject->methodCount(); ++i)
           {
               QString name =  QString::fromLatin1(metaObject->method(i).signature());
               std::cout << name.toStdString() << endl;
               files.write((name + "\n").toAscii());                           } 20
               QMetaMethod method = metaObject->method(i);
               if (metaObject->indexOfSlot(name.toAscii()) != -1) ----------------- 21
                   component->getServer()->attachSlot(compoName+"_"+name, component, SLOT(method));------ 22

           }

           files.close();  --------------------------------------------------------- 24
       }
       return sentenceList; ----------------------------------------------------- 25
```

Fig. 10: Numeración del método *searchConnectionSentence*

```
void ConnectionFactory::connectComponents( QList<AbstractComponent*> compoConnectList1,QList<AbstractComponent*> compoLoadlist) {
QList<AbstractComponent*>compoConnectList=compoConnectList1;
QList<Connect*> connectionList; AbstractComponent* transmitterCompo;          } 1
AbstractComponent* receivingCompo;
 this->setCompoLoadList(compoLoadlist);
        2         3               15
for( int i=0;i< compoConnectList.size();i++) {
 connectionList.clear();                                                        } 4
 connectionList= buildConnection(compoConnectList.at(i)->getNameOfComponent());
   if (!connectionList.isEmpty()) { ---------------------------------------------5
            6      7              14
      for (int j=0;j<connectionList.size(); j++) {
       transmitterCompo=NULL;
       receivingCompo=NULL;                                                     } 8
       transmitterCompo=objComponentManage->searchComponent(compoLoadlist,connectionList.at(j)->getTransmitterData()->transmitterName);
       receivingCompo=objComponentManage->searchComponent(compoLoadlist,connectionList.at(j)->getReceivingData()->receivingName);
      if(transmitterCompo !=0) { -------------------------------------------------9
        if(receivingCompo!=0) { ---------------------------------------------------10
        connectionGeneralList.append(connectionList.at(j));
        connectionList.at(j)->connectComponents(transmitterCompo,receivingCompo);  } 11
        componentsConnectList.insert(transmitterCompo,receivingCompo);
        }
       else QMessageBox::warning(windowMessage,"Aplicacion","El componente " +connectionList.at(j)->getReceivingData()->receivingName+   ----12
      " no se encuentra cargado, verifique la sentencia de conexion del XML.");
           }
      else QMessageBox::warning(windowMessage,"Aplicacion","El componente " +connectionList.at(j)->getTransmitterData()->transmitterName+  ----13
       " no se encuentra cargado, verifique la sentencia de conexion en el XMl del receptor");
      }
    }
  }
 } -----------------------------------------------------------------------------16
```

Fig. 11: Numeración del método connectComponents

```
void Application::loadComponent(QString name)
{
   AbstractComponent* compo=objComponentManage->loadComponent(name);
   if(!compo==0) {
     if(!objComponentManage->componentExists(componentLoadList,name)) {
        compo->getComponentData()->active=true;
        activeActionsComponent(compo);
        componentLoadList.append(compo);
        qDebug()<<"El componente ha sido cargado correctamente";
     }
     else
      QMessageBox::warning(objComponentView->getMainWindow(),"Aplicacion","Este componente ya esta cargado");
   }
   else
       QMessageBox::warning(objComponentView->getMainWindow(),"Aplicacion","El componente "+name +" no se ha podido cargar");
}
```

Fig. 12: Funcionalidad *loadComponent*

```
AbstractComponent* ComponentManage::searchComponent(QList<AbstractComponent*>list,QString name)
{
    AbstractComponent* compo=NULL;
    for(int i =0; i< list.size();i++) {

        if(list.at(i)->getNameOfComponent()==name) {
           compo=list.at(i);
           break;
        }
    }
    return compo;
}
```

Fig. 13: Funcionalidad *searchComponent*

```
bool ConnectionFactory::connectionExists(QString compoName)
{
    bool Var=false;
    QString transmitter;
    QString receiving;
    QDir direccion= objDataController->searchLocationXml();
    QFile file(direccion.absolutePath()+"/"+compoName+".xml");

    if (!file.open(QIODevice::ReadOnly))
        return "";
    if (!docComp.setContent(&file))
        file.close();
    for(int i=0;i<docComp.childNodes().at(0).toElement().elementsByTagName("Componente").size();i++) {

    transmitter = objDataController->searchConnectionSentenceXml(docComp,"Componente","emite",i);
    receiving = objDataController->searchConnectionSentenceXml(docComp,"Componente","recibe",i);

    if(receiving!="" && transmitter!="") {
        Var=true;
        break;
      }
    }
    return Var;
}
```

Fig. 14: Funcionalidad *connectionExists*

Anexo#4: Casos de prueba

Tabla 29: Caso de prueba para el camino básico No.1 de la funcionalidad *connectComponents*

Caso de prueba para el camino básico No.1
Camino: 1, 2, 3, 16
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en

cuestión se recorre la lista de componentes con conexión, en caso de que no existan componentes con conexión no se establece la comunicación entre componentes.
Entrada: Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado: Los componentes cargados no se comunicarán.
Resultado obtenido: No se logró la comunicación entre los componentes cargados.

Tabla 30: Caso de prueba para el camino básico No.4 de la funcionalidad *connectComponents*

Caso de prueba para el camino básico No.4
Camino: 1, 2, 3, 4, 5, 7, 16
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión y se procede a la construcción de las conexiones del componente. Se verifica que existan conexiones entre componentes, en caso de no existir no se establece la comunicación entre componentes.
Entrada: Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado: Los componentes cargados no se comunicarán.
Resultado obtenido: No se logró la comunicación entre los componentes cargados.

Tabla 31: Caso de prueba para el camino básico No.7 de la funcionalidad *connectComponents*

Caso de prueba para el camino básico No.7
Camino: 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 16
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión y se procede a la construcción de las conexiones del componente. Se verifica que exista conexión, en caso de existir se procede a recorrer la lista de conexiones con el propósito de asignar valores a los componentes emisor y receptor. Se verifica que el componente emisor haya sido cargado, en caso contrario el sistema muestra un mensaje de error y no se establece la comunicación entre componentes.

Entrada: Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado: Los componentes cargados no se comunicarán.
Resultado obtenido: No se logró la comunicación entre los componentes cargados.

Anexo#5: Estándar de codificación utilizado

Los estándares de codificación son importantes porque conducen a una mayor coherencia en el código y a su vez permiten generar códigos más fáciles de entender. A continuación se define el estándar de codificación utilizado para la implementación de las nuevas funcionalidades que se adicionaron al Framework Basado en Componentes sobre Qt.

Sangría:

- Se utilizan cuatro espacios para la sangría.

Declaración de variables:

- Declarar cada variable en una línea separada.
- Evitar los nombres cortos siempre que sea posible.
- Utilizar un carácter como nombre de variable solo para el caso en que la misma se comporte como contador o variable temporal, es decir cuando el propósito de la misma sea muy obvio.
- Declarar una variable solo cuando sea necesario.
- El nombre de las variables comienzan con letra minúscula, aunque cada palabra consecutiva en el nombre de una variable comienza con letra mayúscula.
- Evitar las abreviaturas.
- El nombre de las clases debe comenzar con una letra mayúscula.

Espacios en blanco:

- Usar siempre una línea en blanco.
- Emplear siempre un solo espacio después de una palabra clave y antes de una llave de cierre.

- Para punteros o referencias, siempre utilizar un espacio entre el tipo y el '*' o '&', pero no espacio entre el '*' o '&' y el nombre de la variable.

Delimitadores:

- Como regla base, la llave izquierda debe estar en la misma línea que el inicio de la instrucción.
- Aclarar que existen casos excepcionales de la regla antes mencionada. Los mismos se ponen en práctica a la hora de declarar funciones y clases. Para estos casos la llave de apertura se ubica en una nueva línea separada de la declaración.
- Se utilizan las llaves cuando el cuerpo de una sentencia condicional contiene más de una línea o cuando la declaración de la misma es algo compleja.
- Se utilizan las llaves cuando el cuerpo de una sentencia condicional está vacía.

Paréntesis:

- Se utilizan paréntesis para agrupar expresiones.

Cambio de Declaraciones:

- Las etiquetas de caso están en la misma columna que el interruptor.
- Cada caso debe tener una declaración de la rotura (o retorno) al final o un comentario para indicar que no hay ninguna rotura intencional.

Saltos de Línea:

- Mantener las líneas más cortas de 100 caracteres, por lo que se insertan saltos de ser necesario.
- Las comas van al final de una línea quebrada y los operadores comienzan a principios de la nueva línea.

Excepciones generales:

- Se deben capturar siempre en el nivel más alto del sistema, es decir en la capa de más abstracción, por lo general es la capa de interfaz de usuario.
- Ordene la captura de excepciones (catch) siempre en orden descendente desde la más particular hasta la más genérica.