Universidad de las Ciencias Informáticas
Facultad 6

Título: Plugin para la integración del Gestor de Procesos de Media v2.0 con la aplicación Carbon Coder.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Yanet Blanco Escalante

Tutor(es): Adnan Fuentes Díaz

Junio de 2013

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Yanet Blanco Escalante

_____
Firma del Autor

Ing. Adnan Fuentes Díaz

_____
Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. Adnan Fuentes Díaz

E-mail: afuentesd@uci.cu

Formación académica:

Ingeniero en Ciencias Informáticas, Universidad de Ciencias Informáticas.

Centro Laboral:

Universidad de Ciencias Informáticas. Centro de Desarrollo de Geoinformática y Señales Digitales. Facultad 6.

Autor: Yanet Blanco Escalante

E-mail: yescalante@estudiantes.uci.cu

AGRADECIMIENTOS

A mis abuelos por enseñarme y educarme estos 23 años.

Mi mamá por estar siempre conmigo y apoyarme en todo momento de la carrera.

A mi hermana, gracias por ser la mejor hermana del mundo.

Roxanna por nunca defraudarme y apoyarme en los buenos y malos momentos.

A mi tutor por aguantarme todo este tiempo y estar ahí cuando más lo necesitaba.

A todos los profesores que en algún momento dedicaron un pedacito de su tiempo en aclararme cualquier duda.

A todas mis amistades del apartamento y los que han estado conmigo en los diferentes grupos de la carrera que de alguna manera han influenciado en el resultado de esta investigación.

A toda mi familia en general por demostrar que la familia es lo mejor del mundo.

DEDICATORIA

A mis abuelos por ser mi apoyo, por brindarme su amor, dedicación y entrega.

A mi mamá, a mi hermana, y a roxanna por compartir sus alegrías y tristezas, por comprenderme y apoyarme incondicionalmente.

A mis hermanos pequeños por ser tan cariñosos, por llenarme de satisfacciones y alegría.

A mi tío, a Margarita, a Jorge y a todos los familiares que me han apoyado gracias por estar ahí.

RESUMEN

En la actualidad la utilización de archivos de audio y video se ha convertido en una de las principales vías de comunicación. Con el adelanto de las tecnologías y las necesidades de compatibilidad y conformidad de formatos, se han desarrollado herramientas informáticas que permiten la conversión de un formato a otro. Carbon Coder es una herramienta que brinda una gran variedad de formatos y funcionalidades que se adaptan a las necesidades de muchas empresas que realizan procesamiento audiovisual, logrando así alcanzar una gran aceptación en el mercado de software. La universidad de Ciencias Informáticas cuenta con el producto Sistema Gestor de Procesos de Media que permite gestionar los procesos de media y lograr integración con estructuras de hardware y software existentes en la empresa donde se despliegue dicho producto. De esta manera para llegar a un mercado más amplio, se requiere de un producto capaz de integrarse con servidores de tipo Carbon Coder. Esta investigación consiste en desarrollar un plugin que logre la integración entre el Sistema Gestor de Procesos de Media y la herramienta Carbon Coder. El documento recoge los resultados de la investigación acerca de la aplicación Carbon Coder y las funcionalidades que se pueden explotar de la misma. Se efectúa un análisis detallado de las tecnologías para el desarrollo del plugin y algunas soluciones similares. Conjuntamente se describen las funcionalidades del plugin, se modela la lógica de la implementación a través de los diferentes diagramas realizados y finalmente se realiza un resumen de las pruebas efectuadas.

Palabras claves: Carbon Coder, Sistema Gestor de Procesos de Media, integración, plugin.

TABLA DE CONTENIDO

Introducción 1

Capítulo 1: Fundamentación Teórica de la solución Propuesta 4

1.1 Introducción 4

1.2 Conceptos Asociados 4
1.2.1 Plugin 4
1.2.2 API 4
1.2.3 Códec 5
1.2.4 Formato contenedor 5

1.3 Análisis del Objeto de Estudio 5
1.3.1 Software de transcodificación Carbon Coder 5
1.3.2 Características principales (Harmonic Inc, 2011) 5
1.3.3 Características Funcionales (Harmonic Inc, 2011) 7
1.3.4 Soluciones similares 8

1.4 Tecnologías de Desarrollo 10
1.4.1 Metodología de desarrollo 10
1.4.2 Lenguaje Unificado de Modelado 11
1.4.3 Herramienta de modelado de software 12
1.4.4 Lenguaje de programación 12
1.4.5 Entorno de desarrollo 13
1.4.6 Biblioteca utilizada 15

1.5 Conclusiones 15

Capítulo 2: Descripción y Análisis de la Solución Propuesta 17

2.1 Introducción 17

2.2 Descripción del SGPM y la herramienta Carbon Coder 17

2.3 Modelo del Domino 18

2.4 Requisitos 20
2.4.1 Requisitos funcionales 20
2.4.2 Requisitos no funcionales 23

2.5 Modelo de Casos de Uso 23
2.5.1 Descripción de los Casos de Uso del sistema 24

2.6 Arquitectura de Software 26
2.6.1 Arquitectura basada en componentes 27

Índice

2.6.2 Arquitectura Cliente Servidor 27
2.7 Conclusiones 28
Capítulo 3: Implementación y Prueba 30
3.1 Introducción 30
3.2 Diseño 30
3.2.1 Patrones de Diseño 30
3.2.2 Diagramas de clases del diseño 36
3.3 Modelo de implementación 37
3.3.1 Diagrama de Componentes 37
3.4 Diagrama de Despliegue 38
3.5 Prueba de software 38
3.5.1 Prueba de Caja Blanca 39
3.5.2 Prueba de Integración 48
3.6 Conclusiones Parciales 49
Conclusiones generales 51
Recomendaciones 52
Bibliografía 53

ÍNDICE DE ILUSTRACIÓN Y TABLA

Ilustración 1. Modelo de Dominio 18
Ilustración 2. Comunicación de Carbon Coder con aplicaciones terceras 19
Ilustración 3. Integración SGPM con Carbon Coder 19
Ilustración 4. Diagrama de Casos de Uso 23
Ilustración 5. Clase Proceso 31
Ilustración 6. Relación Service1 y Mediadora 31
Ilustración 7. Clase Nodo_Plataforma_Carbon 32
Ilustración 8. Paquete EjecutorProcesosWindows 33
Ilustración 9. Relación Controlador_Observador_Proceso 34
Ilustración 10. Paquete Puente_Comunicación 35
Ilustración 11. Clase Diseño Plugin_Carbon 36
Ilustración 12. Diagrama de Componentes 37
Ilustración 13. Diagrama de Despliegue 38
Ilustración 14. Grafo de Flujo 39
Ilustración 15. Función Consumir_Proceso 41
Ilustración 16. Grafo de flujo1 42
Ilustración 17. Función Ejecutar 43
Ilustración 18. Función Ejecutar 44
Ilustración 19. Función Ejecutar 44
Ilustración 20. Grafo de Flujo2 45

Tabla 1. Caso de Uso Codificar 24
Tabla 2. Caso de Uso Dar_Estado 25
Tabla 3. Caso de Uso Cancelar_Proceso 26
Tabla 4. Caso de Prueba #1 42
Tabla 5. Caso de Prueba #2 46
Tabla 6. Caso de Prueba #3 46
Tabla 7. Caso de Prueba #4 47
Tabla 8. Caso de Prueba #5 47
Tabla 9. Caso de Prueba #6 48
Tabla 10. Caso de Prueba Integración #1 49
Tabla 11. Caso de Prueba Integración #2 49

INTRODUCCIÓN

Desde el surgimiento de las primeras computadoras el hombre ha tratado de procesar y automatizar la información mediante la informática, ciencia que incluye un conjunto de disciplinas aplicadas en diferentes ramas de la sociedad, que facilitan el trabajo de las personas en diferentes aspectos de la vida diaria tanto profesional como personal.

Con el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) los medios de comunicación y la informática han ido evolucionando, siendo este último, un pilar importante en el desarrollo de los medios de comunicación. Medios como la televisión, sitios web y otras empresas que realizan procesamiento audiovisual para satisfacer sus necesidades de compatibilidad y conformidad de formatos entre otras, utilizan herramientas informáticas para llevar a cabo estas operaciones tales como: ATEME, Zencoder, encoding.com, Panvidea entre otras. Una de estas aplicaciones es Carbon Coder. Desarrollado por la tecnología *Rhozet*, *Harmonic* empresa de clase universal, brinda soluciones de transcodificación de media que ofrecen la fiabilidad, escalabilidad y compatibilidad de los medios de comunicación que las empresas necesitan. Carbon Coder se basa en archivos de software de transcodificación que facilitan la conversión de media a la mayor variedad de adquisición, edición, difusión, Web y formatos móviles disponibles. (Harmonic Inc, 2011)

Con el objetivo de informatizar el país y desarrollar la industria de software se crea la Universidad de Ciencias Informáticas (UCI). Centro de estudios universitarios donde se forma capital humano especializado que a la vez se vinculan en el proceso de producción de software, para lograr así un aporte en la economía nacional. Para el desarrollo de software la universidad cuenta con varios centros de desarrollo entre los que se encuentra el centro de desarrollo Geoinformática y Señales Digitales (GEySED), cuyo objetivo es desarrollar productos, servicios y soluciones informáticas en el campo del procesamiento de señales digitales y la geoinformática. El departamento Señales Digitales perteneciente a este centro cuenta con el producto Sistema Gestor de Procesos de Media (SGPM), creado con el objetivo de gestionar los procesos de media y lograr integración con estructuras de software y hardware existentes en la institución o empresa donde se implante dicho producto.

Numerosas empresas exitosas como *Amazon.com, MTV, BBC News, Discovery Channel, Yahoo!*, Televisa, entre otras que realizan procesamiento de video, poseen servidores de transcodificación de

medias que usan Carbon Coder. El departamento Señales Digitales cuenta con una plataforma de procesamiento de video muy similar pero que enfoca sus esfuerzos a los procesos del departamento. Las grandes empresas en su mayoría utilizan el Carbon Coder porque satisface todas sus necesidades de procesamiento y no acceden a que se le cambie su estilo de trabajo. El producto de software con que cuenta el departamento para estas tareas (SGPM), no satisface los requerimientos de muchas de estas importantes empresas en cuanto a la compatibilidad y conformidad de formatos, impidiendo así las posibilidades de negocios de esta solución en este mercado.
Lo cual conduce al siguiente **problema a resolver:** ¿Cómo controlar los procesos que se ejecutan en la aplicación Carbon Coder desde el Sistema Gestor de Procesos de Media v2.0?

Se define como **objeto de estudio** los procesos que se ejecutan en la aplicación de transcodificación Carbon Coder, dirigiendo el **campo de acción** a los procesos de codificación de video y audio y generación de *Storyboard*[1] y *Thumbnails*[2].

Para dar solución al problema se plantea como **objetivo general** desarrollar un plugin para la integración del Sistema Gestor de Procesos de Media v2.0 con la aplicación Carbon Coder.

Por lo que se define la siguiente **idea a defender:** Con el desarrollo del plugin se controlarán las funcionalidades de transcodificación que brinda el Carbon Coder desde el Sistema Gestor de Procesos de Media v2.0.

Para el cumplimiento del objetivo planteado se definen **tareas de investigación** tales como:

1. Caracterizar las arquitecturas de componentes adicionales y los sistemas de procesamiento de media.

[1] Un storyboard es un conjunto de ilustraciones mostradas en secuencia con el objetivo de servir de guía para entender una historia, pre visualizar una animación o seguir la estructura de una película antes de realizarse o filmarse.
[2] Los thumbnails o miniaturas son versiones de imágenes, usadas para ayudar a su organización y reconocimiento.

2. Caracterizar la herramienta de codificación Carbon Coder, así como su API para manejar y determinar funcionalidades a explotar del mismo.

3. Realizar el diseño e implementación del plugin para la integración del Sistema Gestor de Procesos de Media v2.0 con el Carbon Coder.

4. Realizar pruebas al plugin para la integración del Sistema Gestor de Procesos de Media v2.0 con el Carbon Coder.

Dichas tareas permitirán obtener **posibles resultados**:

- Documentación que recoge la modelación del plugin para la integración del Sistema Gestor de Procesos de Media v2.0 con el Carbon Coder.

- Plugin para la integración del Sistema Gestor de Procesos de Media v2.0 con el Carbon Coder.

- Documentación sobre las pruebas realizadas al plugin para la integración del Sistema Gestor de Procesos de Media v2.0 con el Carbon Coder.

Para lograr el cumplimiento de los objetivos planteados se utilizan los siguientes **métodos de investigación**:

Métodos Teóricos

- **Histórico – Lógico:** Para realizar un estudio de las tendencias históricas y actuales en el desarrollo de plugin.

- **Analítico – Sintético:** Para analizar la herramienta de codificación Carbon Coder, así como su API para determinar las funcionalidades a explotar y definir cómo integrarlas al Sistema Gestor de Procesos de Media v2.0. Valorar y analizar las herramientas y tecnologías para el desarrollo del objetivo planteado.

- **Modelación:** Para modelar los requerimientos funcionales que se proponen en el análisis y diseño del plugin para la integración del Sistema Gestor de Procesos de Media v2.0 con el Carbon Coder.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se especifican varios de los conceptos esenciales que permitirán un mejor entendimiento del dominio del problema. De igual forma se realiza un análisis del objeto de estudio para definir la aplicación Carbon Coder y puntualizar las principales características de la misma. Finalmente se realiza un estudio de las soluciones existentes, y se describe e identifican las tecnologías que se van a utilizar para el cumplimiento del objetivo trazado.

1.2 Conceptos Asociados

Para entender la solución propuesta es necesario tener un previo conocimiento de un conjunto de conceptos que proporcionarán un mejor entendimiento a lo largo de toda la investigación. En esta sección se abordarán algunas de estas definiciones.

1.2.1 Plugin

Un plugin (o plug-in) es un programa de computador que interactúa con otro programa para aportarle una función o utilidad específica. Este programa adicional es ejecutado por la aplicación principal. Los plugins típicos tienen la función de reproducir determinados formatos de gráficos, datos multimedia, codificar / decodificar e-mails, filtrar imágenes de programas gráficos, entre otros.
En la actualidad su uso es muy común. Se utilizan como una forma de expansión para programas de forma modular, de forma que se puedan añadir nuevas funcionalidades sin afectar a las ya existentes ni complicar el desarrollo del programa (Escobar, y otros, 2006).

1.2.2 API

En el contexto de la programación de computadoras, API es una abreviatura para interfaz de programas de aplicaciones o interfaz de programación de aplicaciones. Una API es un conjunto de funciones para un programa de aplicaciones o un sistema operativo que los programadores consultan desde dentro de los programas que crean. Por ejemplo la API de *Windows* incluye código para los diversos controles de cuadros de diálogos conocidos por todas las personas que emplean una computadora. Las funciones

para desplazarse por las carpetas de archivos es un elemento de la API de Windows que puede ser útil en cualquier programa de aplicaciones que permite a los usuarios abrir o guardar archivos. Las APIs se suelen distribuir como parte de un SDK (Dan, 2008).

1.2.3 Códec

Los códecs son algoritmos utilizados para traducir una señal analógica en un fichero digital lo más compacto posible, y posteriormente reproducir la forma original de la onda con la mayor fidelidad posible. Es decir, algoritmos de codificación y descodificación (Zator Systems, 1990).

1.2.4 Formato contenedor

Se denomina formato contenedor a un formato de archivo que contiene varios tipos de datos (por ejemplo video, sonido y textos), comprimidos mediante una serie de códecs. Algunos ejemplos de formatos contenedores son: AVI, MOV, OGG, MP4, ASF, etc. (Vega, 2011).

1.3 Análisis del Objeto de Estudio

1.3.1 Software de transcodificación Carbon Coder

Carbon Coder es un software de transcodificación que brinda una amplia gama de funcionalidades y una gran variedad de formatos compatibles. Las escalas del sistema que presenta permiten ajustar los requisitos iniciales con gran exactitud además de que brinda un API abierta permitiendo así la creación de flujos de trabajo personalizados (Harmonic Inc, 2011).

Carbon Coder proporciona un alto rendimiento, una transcodificación escalable y rentable para una amplia gama de entornos empresariales, desde estudios especializados hasta instalaciones de escala empresarial (Harmonic Inc, 2011).

1.3.2 Características principales (Harmonic Inc, 2011)

- **Software de transcodificación basado en archivos.**

Utiliza un motor de flujo de trabajo basado en ficheros. Todas sus habilidades se basan en la adquisición, transformación y entrega de archivos.

- **Amplia compatibilidad de formatos**

Es la capacidad de codificar cualquier formato de producción de media virtualmente para cada estándar de media en uso hoy en día.

- **Interfaz de usuario intuitiva**

La interfaz de usuario intuitiva de Carbon Coder le da un control completo sobre todos los aspectos del proceso de transcodificación.

- **Operación automatizada**

Carbon Coder se puede ejecutar en un modo totalmente automático con soporte para procesamiento por lotes, carpetas de inspección y transferencias FTP automáticas. Carbon Coder también proporciona una transcodificación inteligente para aumentar la productividad al permitir la identificación de formatos de origen y formatos de destino y luego transcodificar de forma automática.

- **Transcodificación Escalable**

Para tareas de transcodificación grandes y múltiples Carbon Coder se puede configurar como una granja de transcodificación, bajo el control del software Carbon Server.

- **SDK[3] basado en XML[4]**

Carbon Coder puede ser controlado directamente a través de un SDK basado en XML proporcionado con el software. Todos los aspectos del proceso de transcodificación puede ser controlado por el SDK, incluyendo características de origen / destino, parámetros de transcodificación, filtrado, composición, inserción de anuncios, titulación, notificaciones, etc.

---

[3] kit de desarrollo de software o SDK (siglas en inglés de software development kit). Es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto.

[4] XML: siglas en inglés de eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C).

1.3.3 Características Funcionales (Harmonic Inc, 2011)

- **Códecs de vídeo soportados**

H.263, H.264, VC-1, Flash, DV25, DV50, DVCPro, DPS, DivX, JPEG 2000, DNxHD, AVCIntra, Secuencias de imágenes, MPEG-1, MPEG-2, MPEG-4, Windows Media, RealVideo.

- **Códecs de audio soportados**

AAC, AMR-NB, WM Audio, RealAudio, Dolby Digital, Dolby E, PCM, MPEG-1 Layer II, MP3, Dolby Digital Plus.

- **Formatos contenedores soportados**

HDV, MXF (incluyendo D-10/IMX), MPEG-2 PS, MPEG-2 TS, GXF, LXF, QuickTime, WMV, ASF, AVI, VOB, 3GPP, 3G2, WMA, WAV, Broadcast WAV, Smooth Streaming, (H.264, VC-1), HTTP Live Streaming.

- **Sistemas compatibles**

Leitch VR, Nexio, Grass Valley Profile, K2, Omneon Spectrum, Quantel sQ, Avid Editing, Systems, Apple Final, Cut Pro, Adobe Premiere Pro, Canopus Edius.

- **Operaciones básicas con Vídeo**

Conversión de tamaño de frame, conversión de tasa de frame, conversión del espacio de color, conversión aspect ratio, conversión entrelazado / desentrelazado, inverse telecine, conversión PAL / NTSC, conversión SD / HD, recorte.

- **Procesamiento de vídeo**

Fade In / Out, corrección Negro / Blanco, mancha, corrección de color, corrección gamma, NTSC-safe, mediana, girar, afilar, reducción de ruido temporal.

- **Procesamiento de audio**

Normalizar, Fade In / Out, De paso bajo, volumen, compresor de gama dinámica, low-pass.

1.3.4 Soluciones similares

Harmonic ofrece a sus socios tecnológicos una solución flexible, escalable y asequible que se integra perfectamente en los flujos de trabajo y soluciones existentes (Harmonic Inc, 2011).

- **AuthenTec**

Es un proveedor líder de la telefonía móvil y la seguridad de la red. El producto es diverso y ofrece la ayuda tecnológica para proteger a las personas y organizaciones a través de redes seguras, la protección de contenido y datos, control de acceso y seguridad de huella digital fuerte. La tecnología de encriptación, los sensores de huellas digitales y el software de gestión de identidades de AuthenTec se despliegan por las principales empresas de dispositivo móvil, redes y computación, proveedores de contenidos y de servicios, y los gobiernos de todo el mundo. Los productos y tecnologías de AuthenTec garantizan la seguridad de cientos de millones de dispositivos. Las listas de clientes de primer nivel incluyen a Alcatel-Lucent, Cisco, Fujitsu, HBO, HP, Lenovo, LG, Motorola, Nokia, Orange, Samsung, Sky, e Texas Instruments (AuthenTec, 2012).

AuthenTec se integra con Carbon Coder para habilitar el cifrado de video sobre la marcha en directo y reproducirlo sin problemas en los dispositivos de usuario final. La solución combinada permite a los proveedores y distribuidores de contenidos proteger y ofrecer streaming en directo y bajo demanda de contenido de video directamente a los usuarios de dispositivos móviles (Harmonic Inc, 2011).

- **Audible Magic**

Como el pionero de la industria, Audible Magic es reconocido como el líder de facto en la monetización, la protección, medición y verificación de contenido en todas sus formas, incluidas la radio y la televisión, Internet y satélite, archivos digitales almacenados , dispositivos de consumo y transferencias de archivos a través de la red.

La tecnología sigue con precisión y controla la detección de material con derechos de autor o cualquier otro tipo de audio o de video basados en contenido. Junto con la base de datos única y dinámica de más de 11 millones de huellas digitales, CopySense Audible Magic ® proporciona productos y la tecnología de identificación por derechos de autor sensible que elimina a los ruidos y da seguimiento muy preciso de su contenido con derechos de autor (Audible Magic Corporation, 2012).

Capítulo 1: Fundamentación Teórica

La tecnología ha allanado el camino para una amplia y creciente gama de soluciones de identificación y detección de los medios de publicidad, el cumplimiento, la monetización y gestión, la piratería, el registro de contenido y apoyo en litigio (Audible Magic Corporation, 2012).

La tecnología forense de marcas de agua de Audible Magic está disponible como un plug-in para Carbon Coder. Este plug-in permite la incrustación de marcas de agua invisibles de Audible Magic en cualquier formato de video para luego ser identificados por sus servicios de información (Harmonic Inc, 2011).

- **Dalet**

Dalet soluciones permite a los radiodifusores y profesionales de los medios crear, gestionar y distribuir contenido en los canales de los medios de comunicación tradicionales y nuevos, incluyendo la televisión interactiva, la Web y las redes móviles. (Dalet Digital Media System, 2012)

Dalet combina en un único sistema de gestión de activos de una plataforma robusta y probada con capacidades avanzadas de metadatos, un motor de flujo de trabajo configurable, y un conjunto completo de herramientas creativas y de producción construidas específicamente con un propósito. Este entorno integrado y abierto permite de extremo a extremo la gestión de todas las cadenas de Noticias, Deportes y Programa de contenido, y permite a los usuarios mejorar significativamente la eficiencia, y maximizar el uso y el valor de sus activos. Las soluciones de Dalet se entregan a través de un profesional dedicado y un Departamento de Integración de Servicios para garantizar los más altos estándares posibles (Dalet Digital Media System, 2012).

Los sistemas Dalet se utilizan en todo el mundo por miles de usuarios individuales en cientos de productores de contenidos de televisión y radio, incluidos los organismos públicos de radiodifusión (ABS-CBN, BBC, CBC, DR, France TV, RTBF, RFI, Rusia Today, RSR y TSR, RT Malasia, VOA, WDR), las redes comerciales y los operadores (Antena 3, Canal +, FOX, eTV, Orange, Time Warner cable, Warner Bros., Sirius XM Radio) y las organizaciones gubernamentales (Queensland JAG, Cámara de los Comunes canadiense, The Comisión Europea) (Dalet Digital Media System, 2012).

Las soluciones de media de gestión de activos de Dalet se amplían mediante la integración del software de transcodificación Carbon Coder. Mediante la integración de Rhozet Dalet puede proporcionar compatibilidad con el formato principal para sus clientes (Harmonic Inc, 2011).

Todas las soluciones antes descritas se integran con la aplicación Carbon Coder, y el objetivo de su integración corresponde con el propósito de la integración del Sistema Gestor de Procesos de Media. Esto se debe a que cada una utiliza los procesos que Carbon Coder brinda para satisfacer sus necesidades de procesamiento y añadirle funcionalidades a su producto, que es exactamente lo que se pretende con el plugin para la integración del Sistema Gestor de Procesos de Media. Controlando los procesos que se ejecutan en un servidor de tipo Carbon Coder hará del Sistema Gestor de Procesos de Media un producto más íntegro y escalable.

1.4 Tecnologías de Desarrollo

Para el desarrollo del plugin y el puente de aplicación, ambas objetivo general de presente trabajo, es necesario realizar un estudio minucioso de las tecnologías que se van a emplear para cumplir el objetivo planteado. A continuación se presentan los principales conceptos y características de las tecnologías que se ajustan a la solución que se propone.

1.4.1 Metodología de desarrollo

Las Metodologías de Desarrollo de Software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software (Carrillo, y otros, 2008).

Rational Unified Process (RUP)

El Proceso Unificado es un proceso de desarrollo de software que trae consigo un conjunto de actividades necesarias para transformar los requisitos de un usuario en un software. El Proceso Unificado es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de softwares, para diferentes ramas de áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos (Jacobson, y otros, 2000).

Características Generales de RUP (Jacobson, y otros, 2000)

- Dirigido por Casos de uso: Los casos de uso representan los requisitos funcionales de un sistema, que no son más que las funciones que los usuarios y clientes necesitan o desean que cumpla el sistema; por lo que guían el diseño, implementación y prueba en el proceso de desarrollo de software.

- Centrado en la arquitectura: La arquitectura de un software no es más que la vista del diseño completo del sistema centrado en los casos de uso más significativos; influyendo también en la misma requisitos no funcionales del sistema. De tal forma se puede decir que la arquitectura no es más que un modelo del sistema que permite la evolución del sistema a lo largo de todo proceso de desarrollo hasta la etapa final del proceso.

- Iterativo e Incremental: El desarrollo de un producto de software puede durar desde meses hasta años, por lo que RUP define que el desarrollo de software se divida en iteraciones o pequeños proyectos que a la vez permitan lograr incrementos en el desarrollo del producto hasta llegar al producto final.

"Las características de un proceso dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental son de igual importancia. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración. La eliminación de una de las tres características reduciría drásticamente el valor del Proceso Unificado. Es como un taburete de tres patas. Sin una de ellas, el taburete se cae." dicho por Ivar Jacobson, Grady Booch y James Rumbaugh.

1.4.2 Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML) es, tal como su nombre lo indica, un lenguaje de modelado y no un método o un proceso. El UML está compuesto por una notación muy específica y por las reglas semánticas relacionadas para la construcción de sistemas de software. El UML en sí mismo no prescribe ni aconseja cómo usar esta notación en el proceso de desarrollo o como parte de una metodología de diseño orientada a objetos (Sparks, 2012).

El UML soporta un conjunto rico en elementos de notación gráficos. Describe la notación para clases, componentes, nodos, actividades, flujos de trabajo, casos de uso, objetos, estados y cómo modelar la

relación entre esos elementos. El UML también soporta la idea de extensiones personalizadas a través de elementos estereotipados (Sparks, 2012).

El UML provee beneficios significativos para los ingenieros de software y las organizaciones al ayudarles a construir modelos rigurosos, trazables y mantenibles, que soporten el ciclo de vida de desarrollo de software completo (Sparks, 2012). La versión utilizada en el desarrollo del plugin es la 2.0.

1.4.3 Herramienta de modelado de software

Se puede definir a las Herramientas CASE[5] como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un *Software* (Instituto Nacional de Estadística e Informática).

Visual Paradigm 8.0

Visual Paradigm for UML es una potente plataforma de desarrollo visual compatible con el ciclo completo de una aplicación y combina el modelado con excelentes herramientas para generación de códigos a partir de diagramas realizados e ingeniería inversa[6] soportando lenguajes de programación como C++ y .NET entre otros. Además de ser una herramienta multiplataforma creada para el trabajo tanto en Windows como en Linux; también permite la exportación de modelos en diferentes formatos y generación automática de informes en formato PDF, Word o HTML.

1.4.4 Lenguaje de programación

Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que

---

[5] Computer Aided Software Engineering (Ayuda por Computadora a la Ingeniería de Software).

[6] Ingeniería Inversa: proceso ingenieril en el que se obtienen modelos conceptuales a partir de los artefactos software como código fuente, ejecutables, binarios y ficheros intermedios.

controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación (Fuentes, 2011).

Lenguaje C++

Posee variadas librerías externas que potencian y facilitan el trabajo a los programadores. Posteriormente se añadieron facilidades de programación genérica permitiendo la herencia múltiple, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multi paradigma (Groningen, 2012).

Lenguaje C#

C# es un lenguaje orientado a objetos, el mismo permite que los desarrolladores puedan crear una extensa gama de aplicaciones sólidas y seguras que se ejecutan en el Framework .NET. También se puede utilizar para crear aplicaciones cliente para *Windows* tradicionales, servicios Web XML, aplicaciones cliente-servidor, entre otras (Microsoft, 2012).

La sintaxis de C# simplifica muchas de las complejidades de C++ y, a la vez, ofrece funciones eficaces tales como tipos de valores que aceptan valores NULL, enumeraciones, delegados, métodos anónimos y acceso directo a memoria, que no se encuentran en Java. C# también admite métodos y tipos (Microsoft, 2012).

Como lenguaje orientado a objetos, C# admite los conceptos de encapsulación, herencia y polimorfismo. En C#, una estructura es como una clase sencilla; es un tipo asignado en la pila que puede implementar interfaces pero que no admite la herencia (Microsoft, 2012).

1.4.5 Entorno de desarrollo

Un entorno de desarrollo integrado (en inglés *Integrated Development Environment* o IDE) es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar

código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos (E.U de Ingeniería Técnica Informática de Oviedo, 2012).

Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: un editor de texto, un compilador, un intérprete, unas herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones. Hoy en día los entornos de desarrollo proporcionan un marco de trabajo para la mayoría de los lenguajes de programación existentes en el mercado (por ejemplo C, C++, C#, Java, Python y Visual Basic entre otros). Además es posible que un mismo entorno de desarrollo tenga la posibilidad de utilizar varios lenguajes de programación, como es el caso de Eclipse (E.U de Ingeniería Técnica Informática de Oviedo, 2012).

Qt Creator 4.8

Qt Creator ha sido desarrollado para ser un entorno de desarrollo integrado (IDE) multiplataforma adaptado a las necesidades de los desarrolladores de Qt. Qt Creator se ejecuta en los sistemas operativos de escritorio *Windows* y Linux/X11 (Softpedia, 2001).

Proporciona: (Softpedia, 2001)

- Un editor de código de C++.
- Un diseñador de interfaz de usuario integrado.
- Herramientas de gestión de versiones y proyectos.
- Control de versiones.

Visual Studio 2010

Visual Studio es un conjunto de herramientas de desarrollo basadas en componentes y otras tecnologías para compilar aplicaciones eficaces de alto rendimiento. Además, Visual Studio está optimizado para el diseño, el desarrollo y la implementación en equipo de soluciones empresariales (Microsoft, 2013).

Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, Servicios Web XML entre otras. Visual C# utiliza un entorno de desarrollo integrado (IDE), que habilita el uso compartido de herramientas y facilita la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes utilizan las funciones de .NET Framework, las cuales ofrecen acceso a tecnologías clave para simplificar el desarrollo de aplicaciones web ASP y Servicios Web XML (Microsoft, 2013).

1.4.6 Biblioteca utilizada

CarbonSDK brinda facilidades a los programadores que les permite realizar una amplia gama de tareas a través del API de Carbon Coder. Esta biblioteca ofrece clases y propiedades que se pueden utilizar para la implementación de diferentes programas. En esta investigación se utilizaron varias clases, algunas de estas son:

- **Coder:** Esta clase recoge algunas de las operaciones más comunes de los servicios de carbono en funcionamiento. Algunos ejemplos incluyen el ser capaz de reunir información de versión y licencia. Se necesita una interfaz para trabajar con esta clase y con la interfaz de trabajo - haciendo esto toda la inicialización de la conexión se hace en un solo lugar (Harmonic, 2009).

- **Jobs:** Esta clase contiene una serie de parámetros de los *Jobs*. Hay 3 factores primarios para crear un job: nombre fuente archivo, directorio de destino y el id de un *preset*, con estos tres valores se puede poner en cola un trabajo para el carbono.

- **Preset:** Esta clase contiene una serie de parámetros de los *preset*, que no son más que la configuración para un solo códec. Por ejemplo, se podría tener un *preset* llamado "QuickTime MP4 con H.264 para streaming. Carbon Coder cuenta con cientos de *presets* para simplificar la elección de la configuración de los trabajos de transcodificación.

1.5 Conclusiones

Con el estudio de los conceptos asociados al dominio del problema tratado, las soluciones similares existentes, y las tecnologías de desarrollo; se ha logrado identificar los elementos y las características que se deben tener en cuenta durante el diseño y construcción del plugin para la integración del Sistema

Gestor de Procesos de Media v2.0 con la aplicación Carbon Coder. Por lo que se obtienen las siguientes conclusiones:

- La investigación efectuada a las soluciones existentes de productos que se integran con la aplicación Carbon Coder, permitió concluir que los mismos cuentan con características comunes. Además los objetivos de la integración de estos concuerdan con el propósito planteado en el presente trabajo.
- El uso de las tecnologías de desarrollo que se detallaron en la investigación permitirá la realización de un producto de buena calidad. La metodología a utilizar y el lenguaje de modelado u de programación satisfacen todos los requisitos del producto y se ajustan perfectamente a las características y necesidades del mismo.

CAPÍTULO 2: DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA

2.1 Introducción

En el presente capítulo se efectúa una descripción de las características del Sistema Gestor de Procesos de Media y de la solución propuesta. Además se crea el modelo del dominio que permitirá llegar a un mejor entendimiento del producto al cual se va a integrar la solución propuesta. También se detallan las funcionalidades requeridas para el desarrollo del plugin mediante la descripción de los requisitos a partir de los cuales se conformará el diagrama de caso de uso. Igualmente se especifica la arquitectura del sistema, hito fundamental de la fase de elaboración definida por la metodología RUP.

2.2 Descripción del SGPM y la herramienta Carbon Coder

El Sistema Gestor de Procesos de Media está compuesto por 4 módulos (Suaréz, y otros, 2012):

- **Editor de Flujos:** Se encargará de la creación visual de flujos de trabajos. Generará un fichero escrito bajo el estándar de orquestación de procesos de negocios BPEL que será necesitado por otros módulos.

- **Gestor de Procesos de Media:** Se encargará de recibir las peticiones de ejecución de flujos de trabajos. Es el encargado de mandar a ejecutar todos los procesos de manera racional en cada uno de los servidores de procesamiento con que cuente el sistema. Es quien mantiene toda la información real del estado de los flujos y de los servidores de procesamiento.

- **Plataforma de Codificación e Indexación:** Encargada del procesamiento que se despliega en los servidores dedicados de la granja. Permitirá agregar de manera fácil nuevas funcionalidades de procesamiento a la misma.

- **Monitor de Procesos de Media:** Encargada de visualizar todos los flujos de procesos que se estarán llevando a cabo en el GPM. Permitirá tomar algunas acciones de gestión como son: cancelar, encolar, eliminar y refrescar. Además permitirá realizar búsquedas de flujos y procesos por distintos filtros, así como especificar a qué Gestor se desea conectar. Esta aplicación será

totalmente distribuida por lo que permite realizar sus acciones desde cualquier localización geográfica.

El Sistema Gestor de Procesos de Media posee una arquitectura basada en componentes. Estos componentes son funciones agregadas en forma de plugin, que permiten la incorporación de nuevas funcionalidades. Por lo que la solución al problema planteado inicialmente se puede integrar a este sistema.

2.3 Modelo del Domino

El modelo del dominio como su nombre lo indica, no es más que un modelo que se crea en la fase de inicio de la metodología RUP. El mismo permite tener un conocimiento y comprender desde el punto de vista conceptual el contexto del sistema al cual se integra la solución propuesta. En el modelo se muestra un sistema interesado (cliente o sistema) que solicita un flujo de procesos, este es atendido por el SGPM y el mismo según los servidores disponible delega el o los procesos solicitados al servidor o servidores cuyos procesos son monitorizados por el Monitor (módulo del SGPM: Monitor de Procesos de Media).

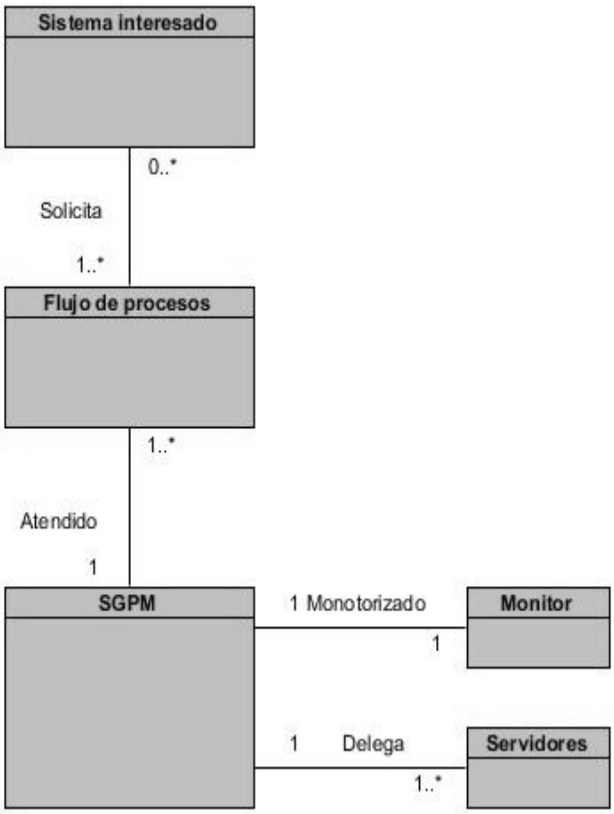

Ilustración 1. Modelo de Dominio

Capítulo 2: Descripción y Análisis de la Solución Propuesta

La herramienta Carbon Coder posee una librería la cual se hizo referencia en el capítulo 1 (CarbonSDK) que permite la comunicación entre aplicaciones terceras y su API, permitiendo la integración de esta herramienta con soluciones existentes. En la siguiente imagen se muestra la comunicación de Carbon Coder con aplicaciones terceras.

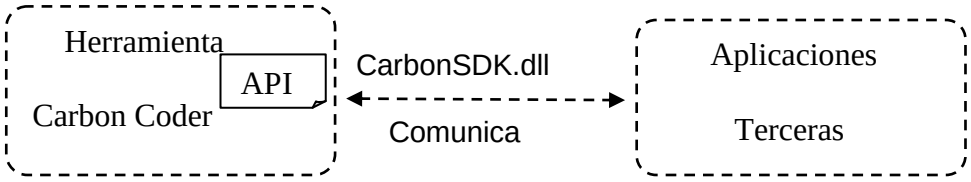

Ilustración 2. Comunicación de Carbon Coder con aplicaciones terceras

El plugin que se va a desarrollar (solución propuesta) tiene como propósito lograr la integración del SGPM con Carbon Coder. Hasta el momento se han contemplado características del SGPM y la herramienta Carbon Coder que demuestran que es posible la integración con estos sistemas. Para alcanzar esto el plugin tiene que lograr comunicarse con el Sistema Gestor de Procesos de Media y el servidor (computadora) donde está instalada la herramienta Carbon Coder.

Para lograr una comunicación de aplicaciones independientes de manera que desde la red se puede acceder a sus funcionalidades, es necesaria la creación de un Servicio Web que no es más que una tecnología basada en estándares que permite la comunicación mediante mensajes sin importar el lenguaje de programación o el sistema operativo. Este servicio web funcionaría como puente entre el plugin y Carbon Coder.

A continuación se muestra un esquema que muestra la integración del SGPM y la herramienta Carbon Coder mediante el plugin a modo general.

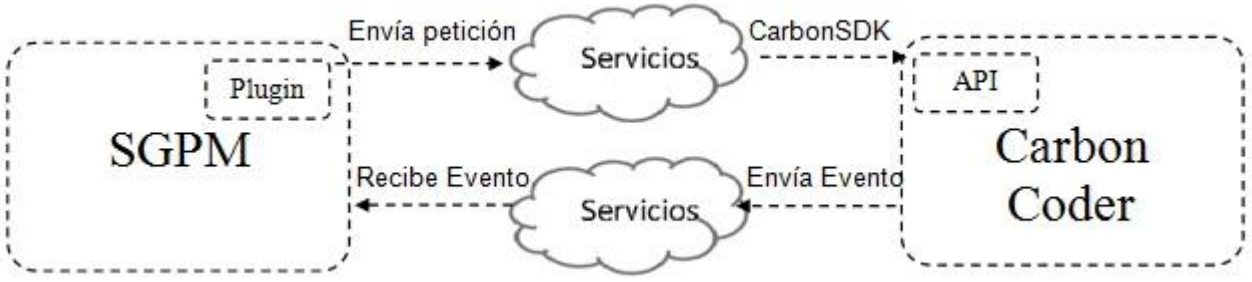

Ilustración 3. Integración SGPM con Carbon Coder

Capítulo 2: Descripción y Análisis de la Solución Propuesta

2.4 Requisitos

Según el Glosario de Terminología Estándar de Ingeniería de Software (IEEE: *Standard Glossary of Software Engineering Terminology*) define al requisito como:

1. Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.
2. Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.
3. Una representación en forma de documento de una condición o capacidad como las expresadas en 1 o en 2.

2.4.1 Requisitos funcionales

RF1: Generar Storyboard.

Descripción: El plugin debe permitir codificar a storyboard un archivo de video del cual se conoce su dirección origen. Este procesamiento digital consiste en convertir un archivo de video en archivos de imágenes (JPEG) por cada intervalo de tiempo definido.

Entradas:

- Identificador.
- Prioridad.
- Nombre.
- Dirección origen del archivo de video.
- Dirección destino de los archivos de imagen.

Salidas:

- Archivos de imagen codificados.

RF2: Generar Thumbnails.

Descripción: El plugin debe permitir codificar a Thumbnails un archivo de video del cual se conoce su dirección origen. Este procesamiento digital consiste en convertir un archivo de video en archivos de imágenes (JPEG) por cada intervalo de tiempo definido.

Entradas:

- Identificador.
- Prioridad.
- Nombre.
- Dirección origen del archivo de video.
- Dirección destino de los archivos de imagen.

Salidas:

- Archivos de imagen codificados.

RF3: Codificar Audio.

Descripción: El plugin debe permitir codificar un archivo de audio del cual se conoce su dirección origen. Este procesamiento digital consiste en codificar un archivo de audio con los parámetros definidos por el usuario.

Entradas:

- Identificador.
- Prioridad.
- Nombre.
- Dirección origen del archivo de video.
- Dirección destino de los archivos de imagen.

Salidas:

- Archivo de audio codificado.

RF4: Codificar Video.

Descripción: El plugin debe permitir codificar un archivo de video del cual se conoce su dirección origen. Este procesamiento digital consiste en codificar un archivo de video con los parámetros definidos por el usuario.

Entradas:

- Identificador.
- Prioridad.
- Nombre.
- Dirección origen del archivo de video.
- Dirección destino de los archivos de imagen.

Salidas:

- Archivo de video codificado.

RF5: Estado de Procesos.

Descripción: El plugin debe ofrecer el estado de los procesos que se ejecutan en la aplicación Carbon Coder.

Entradas:

- Identificador

Salidas:

- Porciento de ejecución del proceso.

RF6: Cancelar Procesos.

Descripción: El plugin debe permitir cancelar los procesos que se ejecutan en la aplicación Carbon Coder.

Entradas:

- Identificador

2.4.2 Requisitos no funcionales

- Requisitos en el diseño e implementación.
  1. Lenguajes de programación C++ y C#.
  2. Como IDE de desarrollo QtCreator 4.8 y Visual Studio 2010.
- Requisitos de software.
  1. Se requiere de la biblioteca CarbonSDK.
- Restricciones de diseño
  1. El plugin será implementado siguiendo el paradigma de la Programación Orientada a Objetos.
  2. Para la modelación del sistema se utilizará como lenguaje de modelado UML.
  3. Se requiere el uso del estilo arquitectónico Cliente-Servidor.
- Eficiencia
  1. El tiempo de respuesta por transacción no debe exceder los 5 segundos.

2.5 Modelo de Casos de Uso

El modelo de Casos de uso no es más que el acuerdo entre el cliente y desarrolladores de los requisitos funcionales que debe cumplir el sistema, también es la base para el desarrollo del análisis y diseño realizados en la fase de construcción del plugin. Un modelo de casos de uso está compuesto por actores, casos de uso y sus relaciones. A continuación se presenta el diagrama de casos de uso del sistema de la solución propuesta.

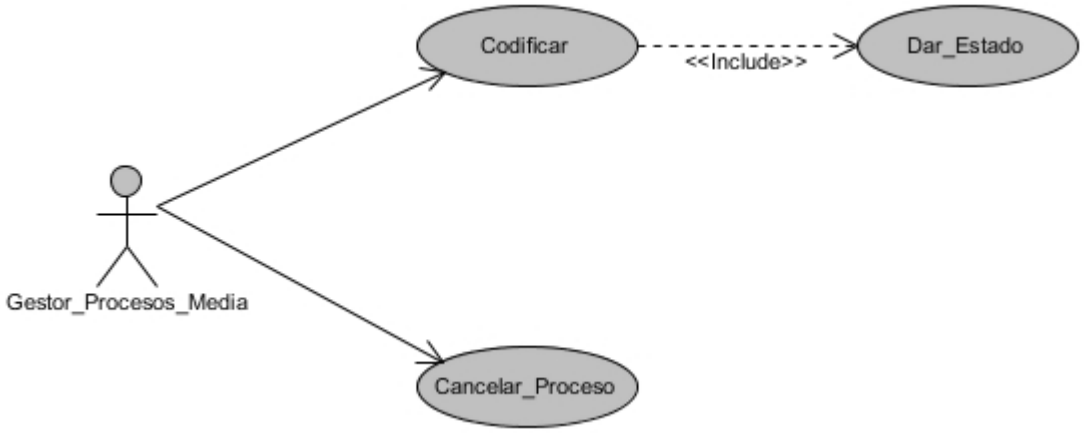

Ilustración 4. Diagrama de Casos de Uso

2.5.1 Descripción de los Casos de Uso del sistema

Caso de Uso:	Codificar
Actores:	Gestor_Procesos_Media
Resumen:	El caso de uso inicia cuando el Gestor delega un flujo de procesos a un servidor de tipo Carbon Coder disponible con los parámetros de entrada del archivo a codificar. El caso de uso finaliza cuando la herramienta Carbon Coder empieza a codificar.
Precondiciones:	Debe existir un servidor disponible.
Referencias	RF1, RF2, RF3, RF4
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El Gestor delega un flujo de procesos al servidor disponible.	1. El Sistema manda a codificar a la herramienta Carbon Coder de acuerdo a los parámetros de entrada recibidos.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2. El Sistema notifica error ocurrido en el proceso.
Poscondiciones	

Tabla 1. Caso de Uso Codificar

Caso de Uso:	Dar_Estado
Actores:	Gestor_Procesos_Media
Resumen:	El caso de uso inicia cuando el Gestor delega un flujo de procesos a un servidor de tipo Carbon Coder disponible y el mismo empieza a codificar. El caso de uso finaliza cuando la herramienta Carbon Coder termina de codificar.
Precondiciones:	Debe existir un proceso de codificación en ejecución.
Referencias	RF5
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El Gestor delega un flujo de procesos al servidor disponible.	1. El Sistema notifica el estado del proceso en porciento del proceso en ejecución.

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2. El Sistema notifica error ocurrido en el proceso.

Poscondiciones	

Tabla 2. Caso de Uso Dar_Estado

Caso de Uso:	Cancelar_Proceso	
Actores:	Gestor_Procesos_Media	
Resumen:	El caso de uso inicia cuando el Gestor envía una petición de cancelar proceso a un servidor de Carbon Coder. El caso de uso finaliza cuando la herramienta Carbon Coder cancela el proceso indicado.	
Precondiciones:	Existir servidor disponible.	
Referencias	RF6	
Prioridad	Crítico	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. El Gestor envía petición de cancelar.		1. El Sistema manda a cancelar el proceso indicado a la herramienta Carbon Coder.
Flujos Alternos		
Acción del Actor		**Respuesta del Sistema**
		2. El Sistema notifica error ocurrido en el proceso.
Poscondiciones		

Tabla 3. Caso de Uso Cancelar_Proceso

2.6 Arquitectura de Software

Según Pressman la arquitectura "… es una descripción de los subsistemas y los componentes de un sistema informático y las relaciones entre ellos (...)."

Capítulo 2: Descripción y Análisis de la Solución Propuesta

Según Jerrold Grochow "… La arquitectura de un sistema constituye un amplio marco que describe su **forma** y su **estructura**, sus **componentes** y cómo estos **interactúan** (...)."

2.6.1 Arquitectura basada en componentes

Primeramente hay que explicar que es un componente ya que es el principal elemento de esta arquitectura.

Según Alan W. Brown y Kurt C. Wallnau definen un componente como "… una parte reemplazable, casi independiente y no trivial de un sistema que cumple una función clara en el contexto de una arquitectura bien definida."

Un componente puede ser un servicio web, un paquete de software o un módulo que encapsula un grupo de funciones relacionadas, los mismos se crean con el objetivo de poder reutilizar sus funcionalidades.

La arquitectura basada en componentes está estructurada por el nombre de los componentes, las interfaces y el código de implementación de dichos componentes. Estas interfaces son las que permiten la comunicación entre componentes, ya que mediante las mismas se puede acceder a las funcionalidades de los componentes sin importar la implementación de estos.

Se utilizó esta arquitectura ya que permite la reutilización de código mediante los componentes y accede a que se le pueda incrementar funcionalidades al sistema fácilmente permitiendo el desarrollo de un producto flexible y escalable.

2.6.2 Arquitectura Cliente Servidor

Los elementos fundamentales de esta arquitectura son el cliente y el servidor por lo que se puntualizan posteriormente para un mejor entendimiento de la arquitectura planteada.

Cliente: Programa ejecutable que participa activamente en el establecimiento de las conexiones. Envía una petición al servidor y se queda esperando por una respuesta. Su tiempo de vida es finito una vez que son servidas sus solicitudes, termina el trabajo (EcuRed, 2010).

Servidor: Es un programa que ofrece un servicio que se puede obtener desde la red. Acepta la petición, realiza el servicio y devuelve el resultado al solicitante. Su tiempo de vida o de interacción es interminable (EcuRed, 2010).

La arquitectura Cliente Servidor se basa en un cliente que realiza peticiones a un servidor que le da respuesta mediante la red, no obstante el cliente y el servidor pueden interactuar desde la misma máquina también. El servidor también puede solicitar una petición a otro servidor y soportan peticiones concurrentes.

Características de la arquitectura Cliente-Servidor (EcuRed, 2010)

- Combinación de un cliente que interactúa con el usuario, y un servidor que interactúa con los recursos a compartir. El proceso del cliente proporciona la interfaz entre el usuario y el resto del sistema. El proceso del servidor actúa como un motor de software que maneja recursos compartidos tales como bases de datos, impresoras, Módem, etc.

- Las tareas del cliente y del servidor tienen diferentes requerimientos en cuanto a recursos de cómputo como velocidad del procesador, memoria, velocidad y capacidades del disco e input-output devices.

- Se establece una relación entre procesos distintos, los cuales pueden ser ejecutados en la misma máquina o en máquinas diferentes distribuidas a lo largo de la red.

- La relación establecida puede ser de muchos a uno, en la que un servidor puede dar servicio a muchos clientes, regulando su acceso a los recursos compartidos.

La utilización de esta arquitectura facilita la integración entre sistemas diferentes permitiendo la comunicación entre el plugin y la herramienta Carbon Coder por lo que se ajusta a las necesidades de la solución planteada.

2.7 Conclusiones

En este capítulo se abordaron las características vitales para la realización de la solución planteada, con el objetivo de lograr un producto de alta calidad que satisfaga las necesidades del cliente, además se

generaron los artefactos precisos para el diseño del sistema, por lo que se puede llegar a las siguientes conclusiones:

- La descripción de los requisitos funcionales adecuados garantiza un desarrollo de software con calidad y eficiencia ya que los requisitos funcionales son las funcionalidades requeridas para satisfacer las necesidades del cliente.
- La elaboración del diagrama de casos de uso definido por la metodología RUP permite que en cada fase del desarrollo se tenga un mejor dominio del sistema que se quiere implementar y constituye la base para la realización de los demás artefactos que se generarán en las fases posteriores.
- Las arquitecturas seleccionadas en este capítulo definen la estructura del sistema y se convertirán posteriormente en la base para el diseño y construcción de la solución propuesta.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

3.1 Introducción

En el presente capítulo se abordará el diseño, implementación y las pruebas definidas, los cuales tienen un gran peso en la fase de construcción planteada por la metodología de desarrollo de software RUP. Se exponen los principales artefactos generados en estas fases de elaboración y construcción de la solución planteada, además de los resultados de las pruebas efectuadas como última etapa del desarrollo de la solución trazada.

3.2 Diseño

El diseño es muy importante ya que es el punto de partida para la implementación. El mismo permite mediante diagramas obtener una visión de forma abstracta de la implementación del sistema permitiendo a otros desarrolladores entender el sistema utilizando una notación común y simple. También se puede usar para la generación de código con las herramientas apropiadas.

El diseño es una representación significativa de ingeniería de algo que se va a construir. Se puede hacer el seguimiento basándose en los requisitos del cliente (Pressman, 2008).

3.2.1 Patrones de Diseño

Los patrones de diseño son soluciones bien documentadas que los desarrolladores emplean para dar solución a nuevos problemas apoyados en la experiencia de haberlas utilizado con éxito en el pasado. Los profesionales identifican partes de un problema que son análogos a otros problemas que han resuelto anteriormente. Luego, retoman la solución utilizada y la generalizan. Por último, adecúan la solución general al contexto de su problema actual (Adriana Sandra Almeira, 2012).

Patrones generales de software para asignar responsabilidades (GRASP)

GRASP son patrones de software para asignación de responsabilidades, es el acrónimo de "GRASP (General Responsibility Assignment Software Patterns)".

A continuación una breve síntesis de los patrones GRASP utilizados:

- **Experto:** Se basa en asignar una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad (Larman, 2005). Un ejemplo donde se evidencia el uso de este patrón es en la clase Proceso mostrada en la siguiente ilustración.

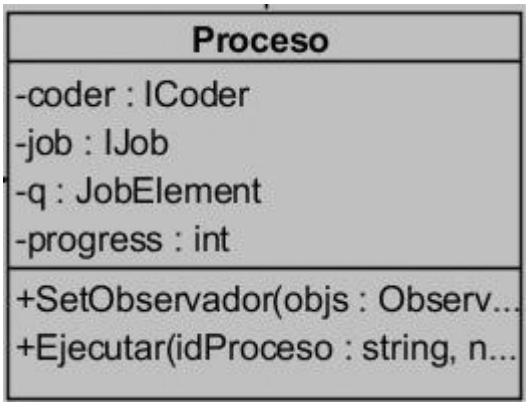

Ilustración 5. Clase Proceso

- **Creador:** Es el responsable de asignarle a la clase B la responsabilidad de crear una instancia de clase A, siempre y cuando B contiene, agrega, utiliza o tenga los datos de inicialización de A. Este patrón es usado en casi todas las clases de la solución propuesta (Larman, 2005). A continuación se muestra una ilustración donde la clase Service1 contiene un objeto de la clase Mediadora, en la cual se evidencia el uso del patrón.

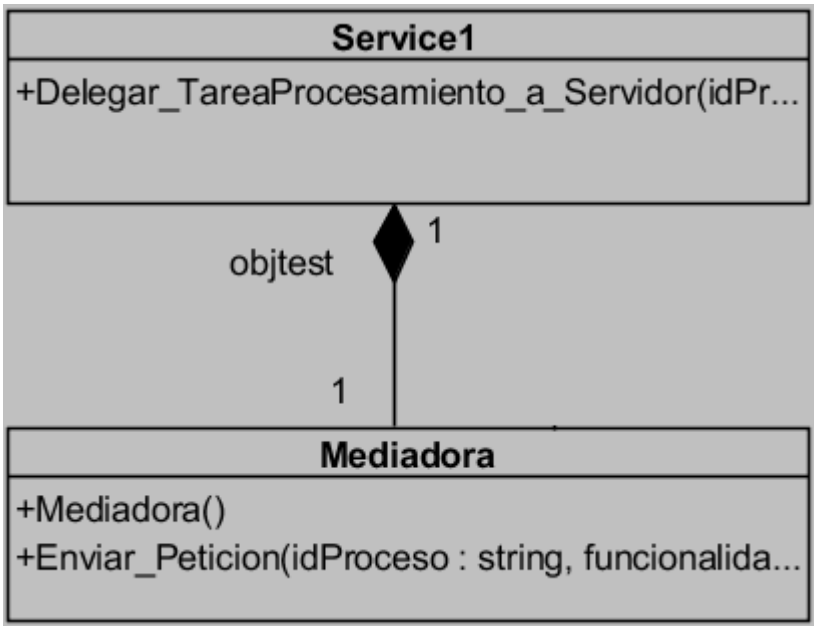

Ilustración 6. Relación Service1 y Mediadora

- **Controlador**: Asignar la responsabilidad de administrar un mensaje de eventos del sistema a una clase que represente una de las siguientes opciones: (Larman, 2005)

    1. El negocio o la organización global (un controlador de fachada).
    2. El "sistema" global (un controlador de fachada).

En la siguiente ilustración de la clase Nodo_Plataforma_Carbon se evidencia el uso de este patrón.

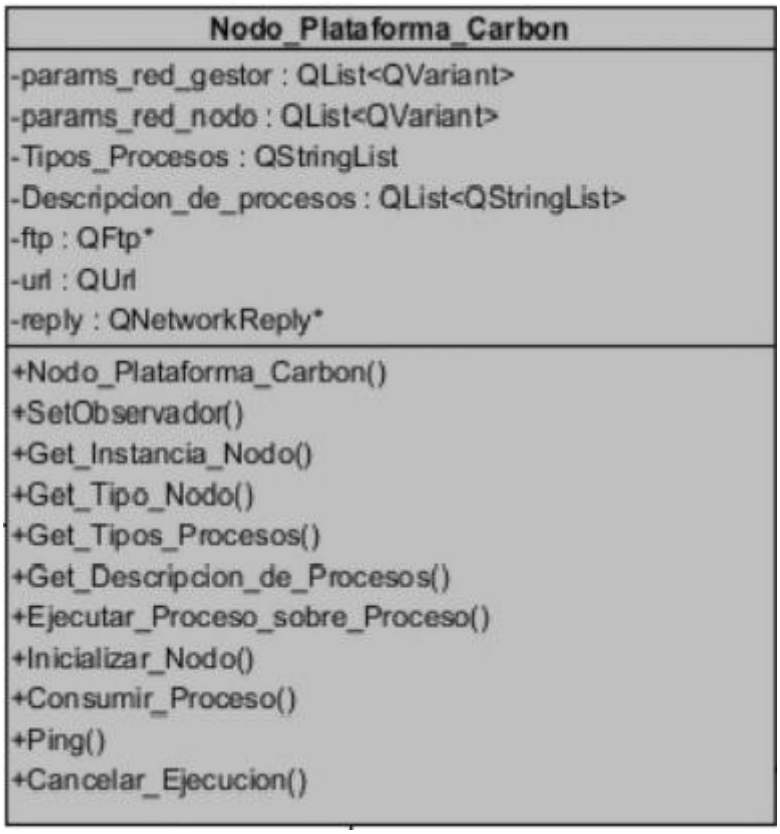

Ilustración 7. Clase Nodo_Plataforma_Carbon

- **Alta Cohesión**: Responde a la siguiente interrogante: ¿Cómo mantener controlable la complejidad? Asigna una responsabilidad de forma tal que la cohesión siga siendo alta, asumiendo como principal objetivo que cada elemento debe de realizar una labor única dentro del plugin. Facilitando la claridad con que se comprende el diseño y una mayor capacidad de reutilización (Larman, 2005).

En la siguiente ilustración se muestra un paquete que contiene las clases EjecutorProcesos, ThreadSample y Proceso donde se evidencia el uso de este patrón ya que se muestra la contribución de las clases mencionadas para realizar tareas de alta complejidad.

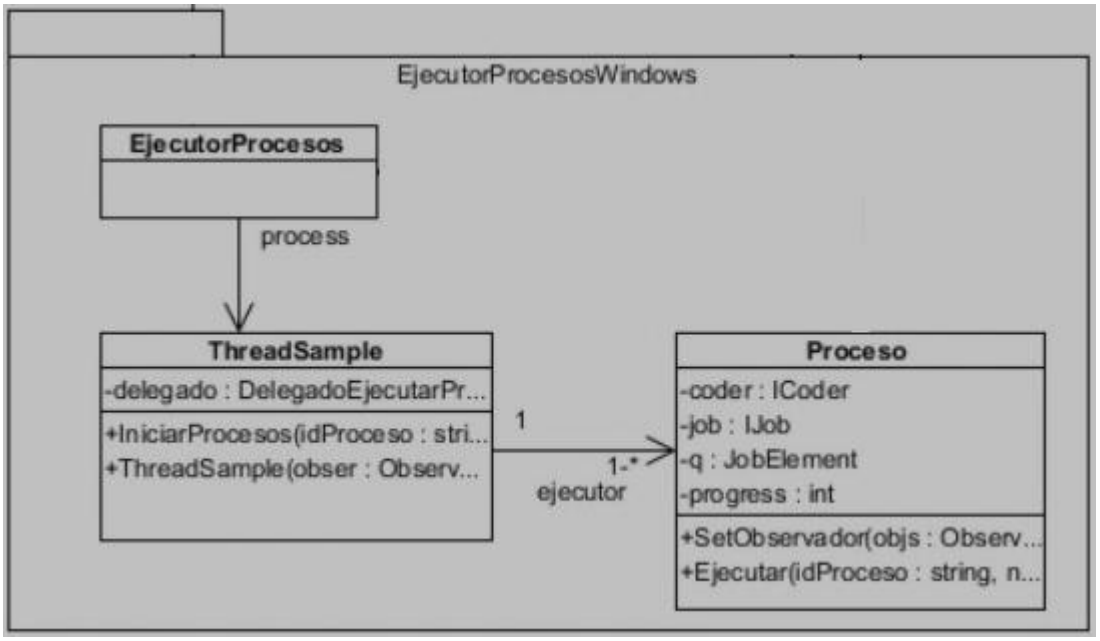

Ilustración 8. Paquete EjecutorProcesosWindows

- **Bajo Acoplamiento:** Responde a la siguiente interrogante: ¿Cómo dar soporte a poca dependencia y a una mayor reutilización? Debe haber pocas dependencias entre las clases. Si todas las clases dependen de todas no se puede extraer software de forma independiente y reutilizarlo (Larman, 2005).

El uso de la arquitectura basada en componentes favorece el uso de este patrón ya que se basa en componentes que se integran con otros componentes independientemente de su implementación, permitiendo la reutilización de los mismos. Por lo que al usar esta arquitectura se evidencia en casi todas las clases su utilización.

Patrones GOF

Los patrones GOF son un conjunto de 23 patrones publicados en el libro Design Patterns escrito por el grupo Gang of Four (GoF) compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, conocidos también como los patrones de la pandilla de los cuatro. Se clasifican según su propósito en patrones creacionales, estructurales y de comportamiento. A continuación una breve descripción del patrón estructural fachada y el patrón de comportamiento observador, patrones utilizados en la solución propuesta.

- **Observador:** Define una dependencia del tipo uno a muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.

En la siguiente ilustración se muestran las clases Controlador (observador) y Procesos (objeto observado) donde se evidencia el uso de este patrón.

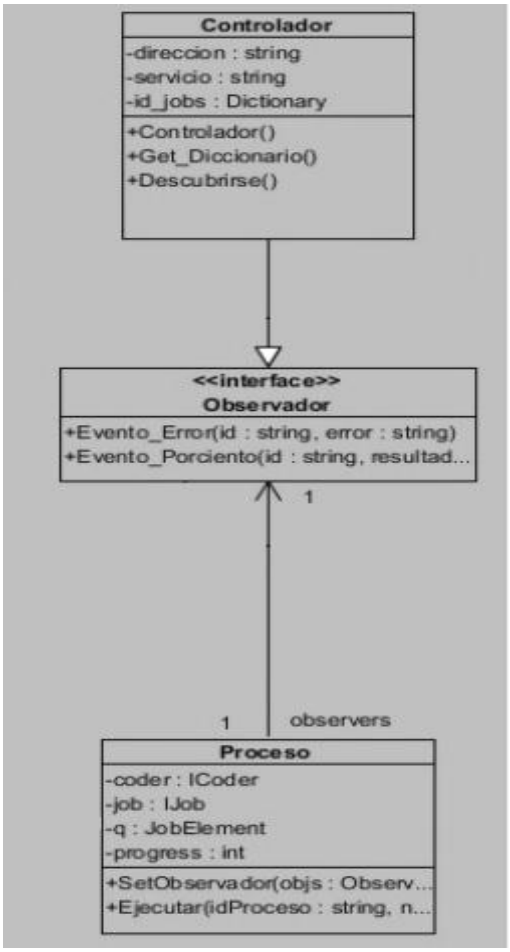

Ilustración 9. Relación Controlador_Observador_Proceso

- **Fachada:** Este patrón se basa en la creación de interfaces para la comunicación entre subsistemas permitiendo el acceso a estos subsistemas mediante la interfaz. Este patrón se evidencia en casi todas las clases ya que se utilizó la arquitectura basada en componentes que implementa este patrón, ya que para la integración entre los componentes del sistema se utilizaron interfaces.

A continuación se muestra un diagrama de componentes donde se evidencia el uso de estas interfaces.

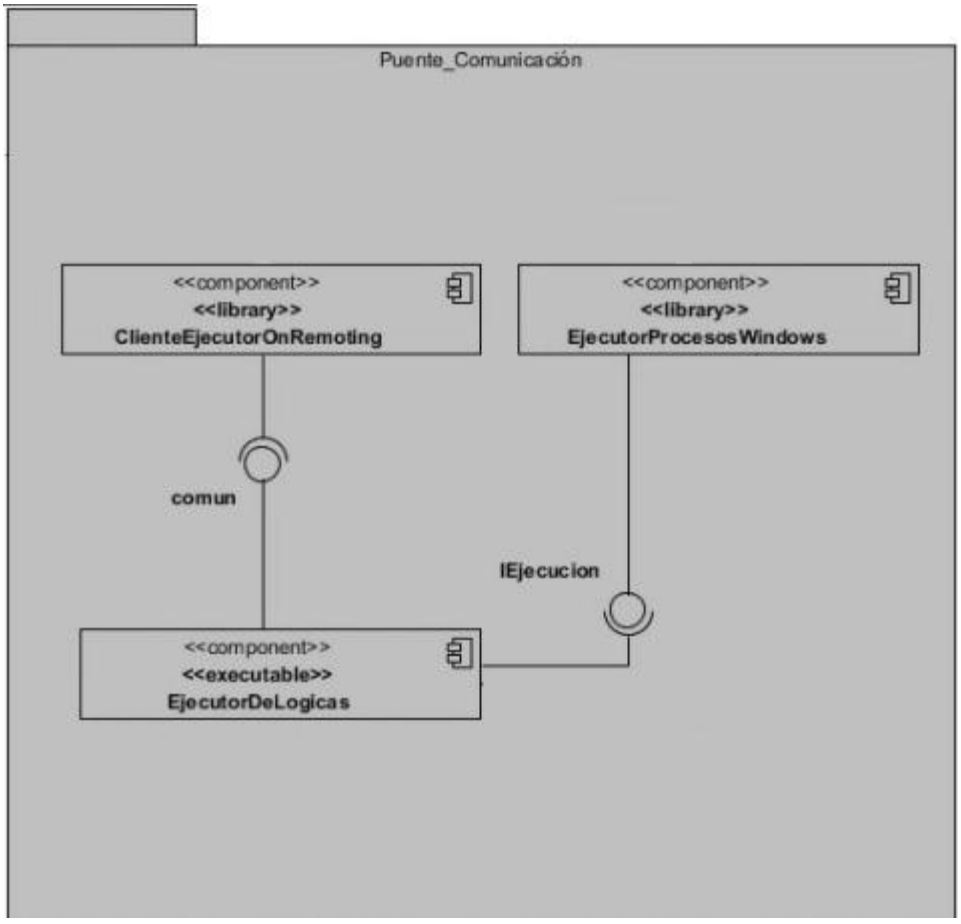

Ilustración 10. Paquete Puente_Comunicación

3.2.2 Diagramas de clases del diseño

Los diagramas de clases se utilizan durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

Un diagrama de clases permite identificar las clases y diseñar el comportamiento, las relaciones, y los datos necesarios para realizar las funciones definidas en las mismas. En la siguiente ilustración se muestra el diagrama de clases del diseño del sistema para representar la relación de las clases de forma visual.

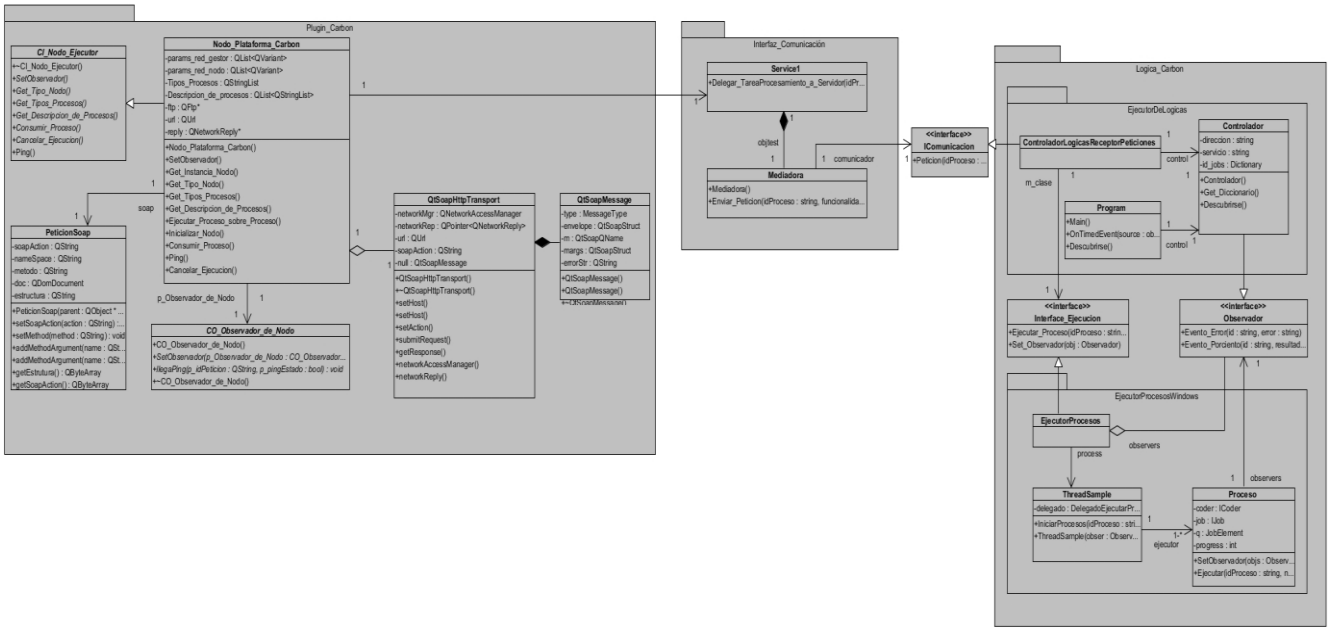

Ilustración 11. Clase Diseño Plugin_Carbon

3.3 Modelo de implementación

El modelo de implementación describe desde el punto de vista estructural y organizativo la forma en que se implementa el sistema en términos de componentes que pueden ser ficheros de código fuente, ejecutables, entre otros, así como la dependencia que existente entre estos componentes.

3.3.1 Diagrama de Componentes

El diagrama de componentes permite describir los elementos físicos del sistema en componentes y sus relaciones. Estos componentes pueden ser interfaces, código binario, bibliotecas cargadas dinámicamente entre otros. A continuación se muestra el diagrama de componentes de la solución propuesta que muestra la relación entre los componentes del sistema agrupados en paquetes.

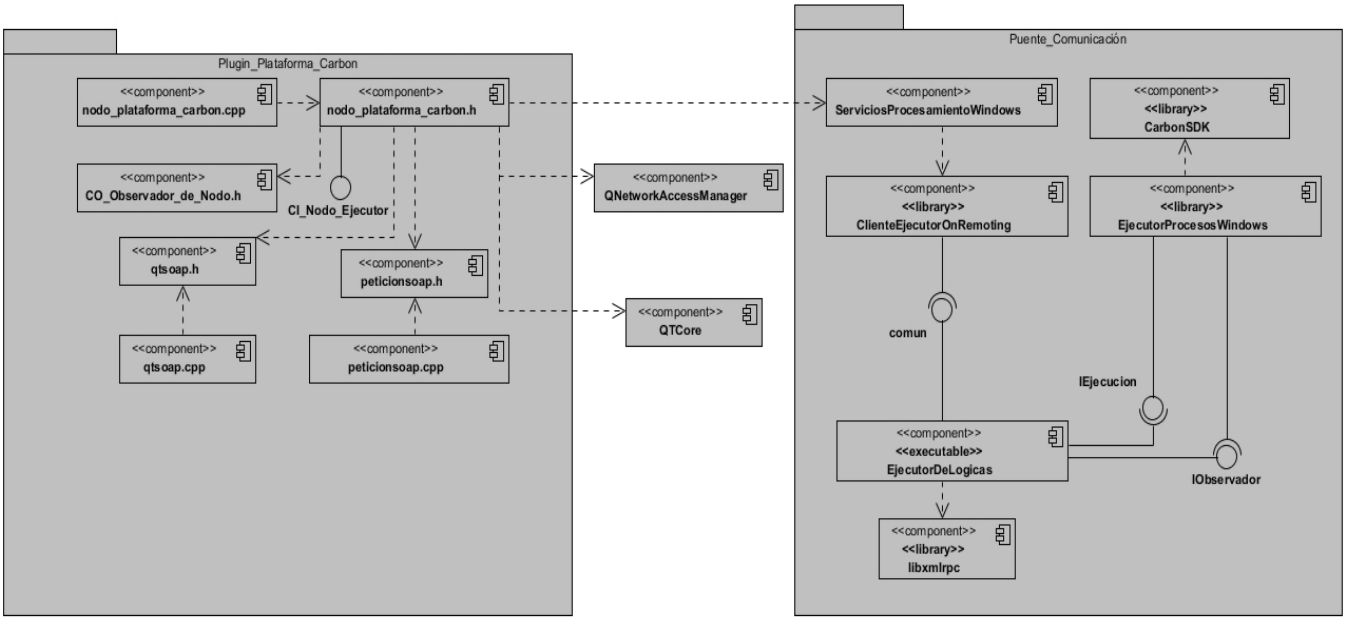

Ilustración 12. Diagrama de Componentes

3.4 Diagrama de Despliegue

El diagrama de despliegue permite modelar mediante nodos las relaciones físicas de los componentes hardware y software usados en la implementación del sistema. A continuación el diagrama de despliegue de la solución propuesta.

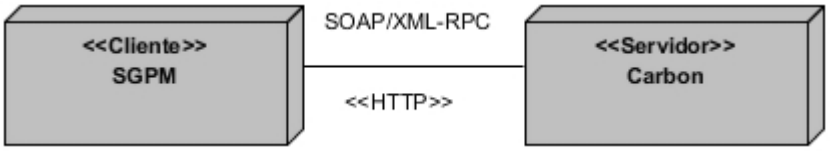

Ilustración 13. Diagrama de Despliegue

En el nodo **SGPM** es donde va a estar desplegado el plugin, el mismo envía una petición SOAP[7] al nodo **Carbon** mediante el protocolo HTTP[8] con los parámetros de entrada. El nodo **Carbon** ejecuta un proceso de acuerdo a la funcionalidad enviada por parámetro y le envía al nodo **SGPM** una petición XML-RPC[9] notificando un evento (estado del proceso o error) mediante el protocolo HTTP.

3.5 Prueba de software

La prueba es un conjunto de actividades que se planean con anticipación y se realizan de forma sistemática. El software se prueba para descubrir errores cometidos sin darse cuenta al realizar su diseño y construcción. Las pruebas son el último bastión para la evaluación de la calidad (Pressman, 2008).

---

[7] SOAP (siglas de Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

[8] Hypertext Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto) es el protocolo usado en cada transacción de la World Wide Web.

[9] XML-RPC es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos.

Capítulo 3:Implementación y Prueba

Según Glen Myers en su libro sobre las pruebas del software, menciona: “La prueba es el proceso de ejecución de un programa con la intención de descubrir un error”.

Para garantizar la calidad y el funcionamiento del producto final se decide realizar las pruebas de Caja Blanca e Integración, las cuales se describen a continuación.

3.5.1 Prueba de Caja Blanca

La prueba de caja blanca, denominada a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero de software puede obtener casos de prueba que primeramente garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo; que ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; luego ejecuten todos los bucles en sus límites y con sus límites operacionales; y finalmente ejerciten las estructuras internas de datos para asegurar su validez (Pressman, 2008).

Para la realización de esta prueba se utilizó la técnica del camino básico. La misma consiste en diseñar casos de pruebas que permitan adquirir una medida de la complejidad lógica de una determinado diseño o código fuente y usar esta medida para obtener un conjunto de caminos básicos que garanticen que durante las pruebas se ejecuta al menos una vez cada sentencia.

Los pasos que se siguen para aplicar esta técnica son:

1. Dibujar el grafo de flujo, a partir del diseño o del código fuente.
2. Calcular la Complejidad Ciclomática del grafo.
3. Determinar un conjunto básico de caminos independientes.
4. Diseñar los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Un grafo de flujo está compuesto por círculos denominados nodos del grafo. Cada nodo representa una o más sentencias. Las flechas se denominan aristas o enlaces y representan el flujo de control. Cada arista debe terminar en un nodo. Las áreas determinadas por los nodos y las aristas se denominan regiones. El

área exterior del grafo cuenta como una región más. A continuación se muestra una ilustración de un grafo de flujo y sus componentes.

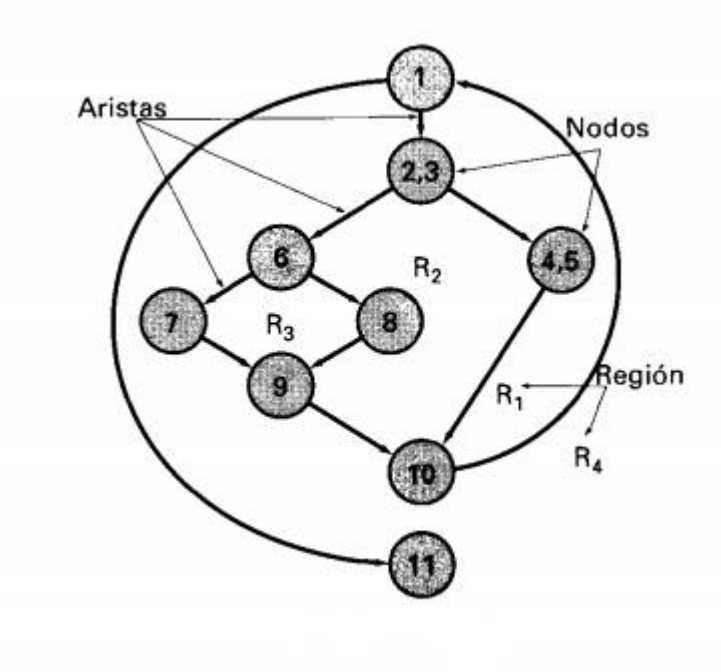

Ilustración 14. Grafo de Flujo

La Complejidad Ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto del método de prueba del camino básico, el valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez (Pressman, 2008).

La complejidad se puede calcular de tres formas:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática.

2. La complejidad ciclomática, V(G), de un grafo de flujo G se define como: V(G)=A- N+2 donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.

3. La complejidad ciclomática, V(G), de un grafo de flujo G también se define como: V(G) =P+ 1 donde P es el número de nodos predicado contenidos en el grafo de flujo G.

Se decidió utilizar la segunda forma para calcular la complejidad ciclomática.

Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición. En términos del grafo de flujo, un camino independiente está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino (Pressman, 2008).

Pruebas de Caja Blanca aplicadas al sistema

- Clase Nodo_Plataforma_Carbon. Función Consumir_Proceso.

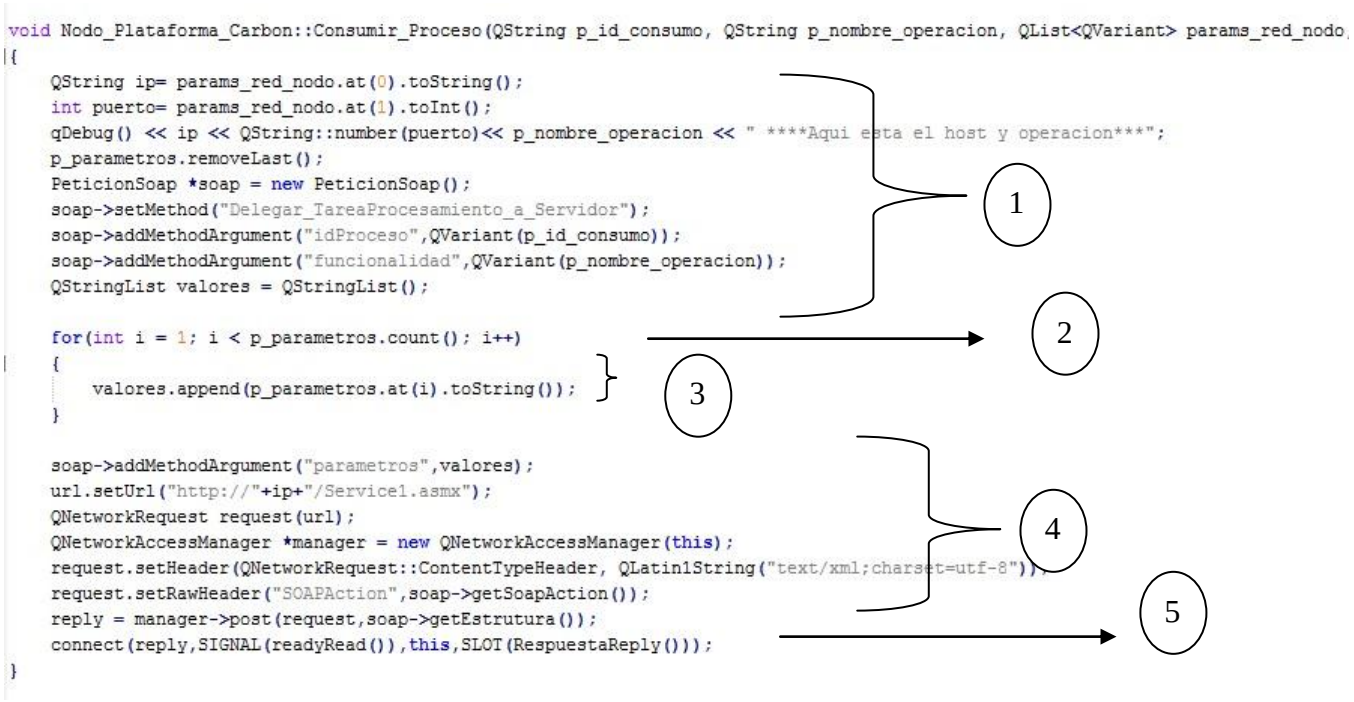

Ilustración 15. Función Consumir_Proceso

Grafo de flujos

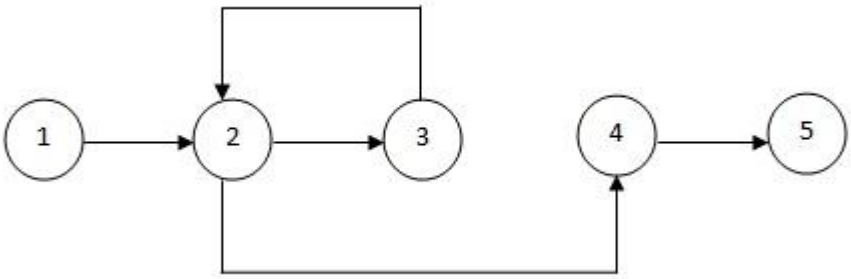

Ilustración 16. Grafo de flujo1

Complejidad ciclomática

V (G) =Cantidad Aristas - Cantidad Nodos + 2

V (G) = 5-5 + 2

V (G) = 2

Caminos básicos

CB1: 1, 2, 3, 2, 4, 5

CB2: 1, 2, 4, 5

Tabla 4.Caso de Prueba #1

Caso de Prueba #1	
Camino :	CB1
Caso de Prueba	Probando Función Consumir_Proceso
Entrada:	Para un valor de: p_id_consumo= 87ds5a, p_nombre_operacion = Codificar_VideoV, p_parametros = AED21E57D6653FF, 10, ved, C:\Users\Yanet\Desktop\Nuevo\v1.avi,

	C:\Users\Yanet\Desktop\test
Resultado Esperado:	Envía a codificar un video al Servidor *Carbon*.
Resultado Prueba:	Satisfactorio

- Clase Proceso. Función Ejecutar

```
public void Ejecutar(string idProceso, string funcionalidad, string[] parametros
{
    try
    {
1    if (funcionalidad == "Codificar")
     {
2        if (File.Exists(parametros[3]) == false)
3            throw new FileLoadException();
4        if (Directory.Exists(parametros[4]) == false)
5            throw new Exception("La dirección destino no existe");
6        if (parametros.Length != 0)
         {
7            q.guid = parametros[0];
             q.priority = int.Parse(parametros[1]);
             q.writeName.Add(parametros[2]);
             q.output = parametros[4];
             q.filename.Add(parametros[3]);
             q.stitch = true;
             job.JobQueue(ref q);

             Jobs.JobStatusElement jsl;
             observers.Get_Diccionario().Add(idProceso, q.jobGUID);

8            do
             {
9                jsl = job.JobStatus(q.jobGUID);
10               if (jsl.State == "NEX_JOB_ERROR")
                 {
11                   observers.Evento_Error(idProceso, "JOB error inesperado,
                     break;
                 }
```

Ilustración 17. Función Ejecutar

```
12              if (progress != jsl.Progress)
13              {
                  progress = jsl.Progress;
                  observers.Evento_Porciento(idProceso, progress.ToString());
                }

14              Thread.Sleep(1000);
            }
15,16        while (jsl.State != "NEX_JOB_COMPLETED" && jsl.State != "NEX_JOB_ERROR");
17           observers.Get_Diccionario().Remove(idProceso);

        }
    }
18  else if (funcionalidad == "Cancelar")
    {
19      if (observers.Get_Diccionario().ContainsKey(idProceso))
        {
20          observers.Get_Diccionario().Remove(idProceso);
        }
        else
        {
21          observers.Evento_Error(idProceso, "No existe el id proceso a cancelar");
        }
    }
22
}
```

Ilustración 18. Función Ejecutar

```
    catch (FileLoadException e)
    {
        observers.Evento_Error(idProceso, "EL fichero a codificar no existe o la dirección es incorrecta");
    }
    catch (Exception e)
    {
        observers.Evento_Error(idProceso, e.Message);
    }
} 23
```

Ilustración 19. Función Ejecutar

Grafo de flujos

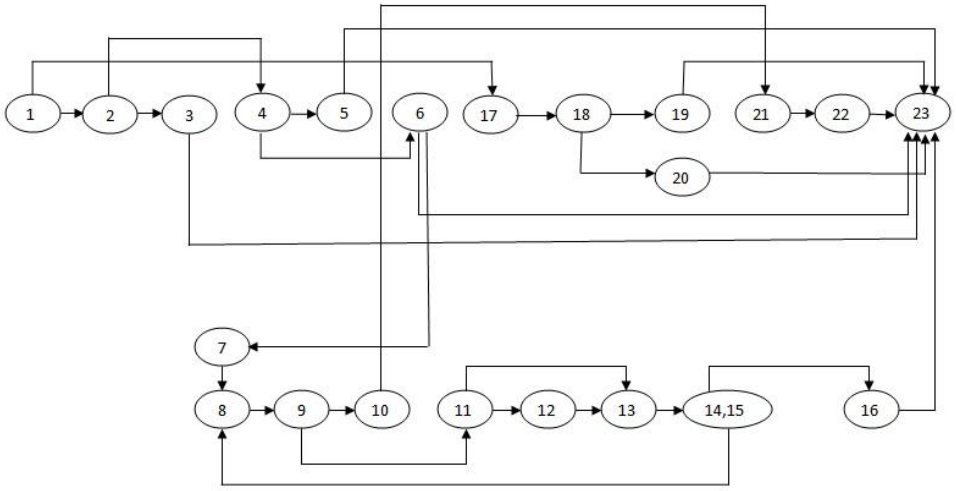

Ilustración 20. Grafo de Flujo2

Complejidad ciclomática

V (G) = Cantidad Aristas - Cantidad Nodos + 2

V (G) = 29-23+2

V (G) = 8

Caminos básicos

CB1: 1, 2, 3, 23

CB2: 1, 2, 4, 5, 23

CB3: 1, 2, 4, 6, 23

CB4: 1, 2, 4, 6, 7, 8, 9, 10, 21, 22, 23

CB5: 1, 2, 4, 6, 7, 8, 9, 11, 13, 14, 15, 16, 23

CB6: 1, 2, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 23

CB7: 1, 17, 18, 19, 23

CB8: 1, 17, 18, 20, 23

Caso de Prueba #2	
Camino :	CB1
Caso de Prueba	Probando Función Ejecutar
Entrada:	Para un valor de: p_id_consumo= 87ds5a, p_nombre_operacion = Codificar_VideoV, p_parametros = AED21E57D6653FF, 10, ved, C:\Users\Yanet\Desktop\Nuevo\v1.av, C:\Users\Yanet\Desktop\test
Resultado Esperado:	Se envía un evento de error.
Resultado Prueba:	Satisfactorio,

Tabla 5. Caso de Prueba #2

Caso de Prueba #3	
Camino :	CB2
Caso de Prueba	Probando Función Ejecutar
Entrada:	Para un valor de: p_id_consumo= 87ds5a, p_nombre_operacion = Codificar_VideoV, p_parametros = AED21E57D6653FF, 10, ved, C:\Users\Yanet\Desktop\Nuevo\v1.avi, P:\Users\Yanet\Desktop\test
Resultado Esperado:	Se envía un mensaje de error.
Resultado Prueba:	Satisfactorio

Tabla 6. Caso de Prueba #3

Caso de Prueba #4	
Camino :	CB6
Caso de Prueba	Probando Función Ejecutar
Entrada:	Para un valor de: p_id_consumo= 87ds5a, p_nombre_operacion = Codificar_VideoV, p_parametros = AED21E57D6653FF, 10, ved, C:\Users\Yanet\Desktop\Nuevo\v1.avi, C:\Users\Yanet\Desktop\test
Resultado Esperado:	La herramienta Carbon Coder codifica video y se envía el evento de estado en porciento de ejecución.
Resultado Prueba:	Satisfactorio

Tabla 7. Caso de Prueba #4

Caso de Prueba #5	
Camino :	CB7
Caso de Prueba	Probando Función Ejecutar
Entrada:	Para un valor de: p_id_consumo= 87ds5a, p_nombre_operacion = Cancelar, p_parametros = “ ”.
Resultado Esperado:	La herramienta Carbon Coder cancela el proceso especificado.
Resultado Prueba:	Satisfactorio

Tabla 8. Caso de Prueba #5

Caso de Prueba #6	
Camino :	CB8
Caso de Prueba	Probando Función Ejecutar
Entrada:	Para un valor de: p_id_consumo= id_carbon, p_nombre_operacion = Codificar, p_parametros = “ ”.
Resultado Esperado:	Se envía un mensaje de error.
Resultado Prueba:	Satisfactorio

Tabla 9. Caso de Prueba #6

3.5.2 Prueba de Integración

Las pruebas de integración se realizan para verificar que las partes o componentes de un sistema de software una vez combinados y probados funcionen como un todo. Existen dos estrategias de integración incremental: Descendente y Ascendente. Para la realización de la prueba de integración se decide utilizar la estrategia Descendente.

La prueba de integración descendente es un enfoque incremental para la construcción de la arquitectura del software. Los módulos se integran al descender por la jerarquía de control, empezando por el módulo de control principal (programa principal) (Pressman, 2008).

Caso de Prueba #1	
Caso de Uso :	Codificar
Entrada:	Para un valor de: p_id_consumo= 87ds5a, p_nombre_operacion = Codificar_VideoV, p_parametros = AED21E57D6653FF, 10, ved, C:\Users\Yanet\Desktop\Nuevo\v1.avi, C:\Users\Yanet\Desktop\test

Resultado Esperado:	La herramienta Carbon Coder codifica video y se envía el evento de estado en porciento de ejecución.
Resultado Prueba:	Satisfactorio

Tabla 10. Caso de Prueba Integración #1

Caso de Prueba #2	
Caso de Uso :	Cancelar_Proceso
Entrada:	Para un valor de: p_id_consumo= 87ds5a, p_nombre_operacion = Codificar_VideoV, p_parametros = “ ”.
Resultado Esperado:	La herramienta Carbon Coder cancela el proceso especificado.
Resultado Prueba:	Satisfactorio

Tabla 11. Caso de Prueba Integración #2

3.6 Conclusiones Parciales

En el presente capítulo se abordaron aspectos valiosos para la realización de la solución planteada, con el propósito de lograr un producto que cumpla con los requisitos funcionales planteados. Además se generaron los artefactos precisos para el desarrollo del sistema, por lo que se puede llegar a las siguientes conclusiones:

- El diseño adecuado garantiza que los programadores tengan una idea de cómo va a estar implementado el sistema en un lenguaje común, convirtiéndose en la base para su desarrollo.
- El uso de los patrones de diseño permitieron lograr una implementación bien estructurada y organizada, logrando así un desarrollo satisfactorio en la construcción de la solución planteada.
- La elaboración de los diagrama de clases, componentes y despliegue definidos por la metodología RUP, permitieron obtener una visión de las relaciones entre los diferentes paquetes del sistema

desde el punto de vista de clases, componentes y nodos que permitieron modelar el comportamiento del sistema desde diferentes perspectivas.

- La ejecución de las pruebas planteadas en este capítulo permitieron verificar que las funcionalidades del plugin se realizaran con la calidad y eficiencia requerida y al mismo tiempo detectar posibles errores ocurridos en la implementación e integración de la solución propuesta.

CONCLUSIONES GENERALES

Una vez concluida todas las tareas de investigación se puede afirmar que se cumplió con el objetivo propuesto en la presente investigación por lo que se procede a las siguientes conclusiones:

El estudio de la herramienta Carbon Coder y las soluciones existentes permitieron alcanzar una fundamentación teórica del objeto de estudio sólida que permitió junto con las tecnologías de desarrollo seleccionadas crear los cimientos para el desarrollo de la investigación. Los artefactos generados por la metodología RUP, la arquitectura seleccionada y las pruebas de software realizadas permitieron desarrollar un producto capaz de cumplir con los requisitos funcionales propuestos por el cliente, logrando de esta manera un plugin capaz de integrar al Sistema Gestor de Procesos de media con la herramienta Carbon Coder. La utilización de este plugin permitirá al SGPM ejecutar procesos de codificación en servidores de tipo Carbon Coder e integrarse con instituciones que utilicen servidores de este tipo.

RECOMENDACIONES

Luego de finalizada la investigación y cumplido con el objetivo de la misma se recomienda incorporar más configuraciones para codificar con Carbon Coder ya que actualmente las que utiliza el plugin son las definidas por el cliente (SGPM). También se sugiere continuar el estudio de la herramienta para controlar más procesos de la misma.

BIBLIOGRAFÍA

Adriana Sandra Almeira, Vanina Perez Cavenago. 2012. eva.uci.cu . *eva.uci.cu* . [En línea] 2012. http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_1/Arquitectura_de_Software/Tesina_Arquitectura_de_Soft.pdf.

Audible Magic Corporation. 2012. Audible Magic. [En línea] 2012. http://www.audiblemagic.com/company.php.

AuthenTec. 2012. AuthenTec. [En línea] 2012. http://www.authentec.com/Company/About.aspx.

AVECO s.r.o. 2012. Aveco. [En línea] 2012. http://www.aveco.com.

Carrillo, Isaías Pérez, Gonzáles, Rodrigo Pérez y Rodríguez, Aureliano David. 2008. Solusoft. [En línea] 15 de 10 de 2008. http://www.solusoft-g11.googlecode.com/.../Metodologias%20de%20....

Corporation, Nokia. 2009. - doc.qt.digia.com -. [En línea] 2009. [Citado el: 20 de 4 de 2013.] http://doc.qt.digia.com/solutions/4/qtsoap.

Dalet Digital Media System. 2012. Dalet. [En línea] 2012. http://www.dalet.com/company-profile.

Dan, Parsons June Jamrich.Oja. 2008. *Conceptos de Computacion: nuevas perspectivas.* 2008.

E.U de Ingeniería Técnica Informática de Oviedo. 2012. Entornos de Desarrollo Integrado. [En línea] 2012. http://www.petra.euitio.uniovi.es/.../Entornos%20de%20Desarrollo%20Integrado.

EcuRed. 2010. EcuRed. [En línea] 14 de 12 de 2010. [Citado el: 20 de 5 de 2013.] http://www.ecured.cu.

Editshare. 2012. Editshare. [En línea] 2012. http://www.editshare.com/index.php?option=com_content&view=article&id=58&Itemid=80.

Bibliografía

Escobar, Claudio Arraigada y Lagos, Joel Llancao. 2006. *'WordPress' y la creación de un sitio Web dinámico: metodología de instalación y puesta en marcha.* Chile A.G : Hector Gómez Fuentes, Director Departamento de Gestión de Información, 2006.

Fuentes, Antonio Norman. 2011. Computación Superior. [En línea] 2011. http://www.computacionsuperior.com.mx/index-2.htm.

Groningen, FRANK BROKKEN B. University of. 2012. *C++ Annotations.* 2012.

Harmonic Inc. 2011. Rhozet: Universal Media Transcoding. [En línea] 2011. http://www.rhozet.com.

—. 2011. Rhozet: Universal Media Trasncoding. [En línea] 2011. http://www.rhozet.com/partners.html.

Harmonic. 2009. *SDK v2 Documentación.* 2009.

Instituto Nacional de Estadística e Informática. Herramientas Case. [En línea] http://www.inei.gob.pe/biblioineipub/bancopub/Inf/.../Libro.pdf.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Educación.S.A, 2000. 84-7829-036-2.

José Carlos Cortizo Pérez, Diego Expósito Gil y Miguel Ruiz Leyva. 2012 . eXtreme Programming. [En línea] 2012 . http://www.esp.uem.es/jccortizo/xp.pdf .

Larman, Craig. 2005. *Applying UML and patterns : An Introduction to object - oriented analysis and design and iterative develop* . s.l. : Prentice Hall, 2005.

Microsoft. 2013. Mirosoft. [En línea] 2013. http://www.microsoft.com/visualstudio/esn/visual-studio-update.

—. 2012. MSND. [En línea] 2012. http://msdn.microsoft.com/es-es/library/z1zx9t92%28v=vs.80%29.aspx.

Bibliografía

MindTouch Deki . 2011. MonoDevelop. [En línea] 2011. http://www.monodevelop.com.

Pressman, Roger S. 2008. *Ingeniería del Software un enfoque práctico.* 2008.

Softpedia. 2001. Softpedia. [En línea] 2001. http://www.visual-paradigm-for-uml-professional-edition.softpedia.com/es/.

—. Softpedia. [En línea] http://www.softpedia.es/programa-Qt-Creator-167814.html.

Sparks, Geoffrey. 2012. Una Introducción al UML. El Modelo de Casos de Usos. [En línea] 2012. http://www.exa.unicen.edu.ar/catedras/modysim/teoria/casos_de_uso_a.pdf.

Suaréz, Jean Michael Pérez, Fuentes, Adnan Díaz y Becerra, Yesleny . 2012. *Arquitectura para Sistema Gestor de Procesos de Media.* La Habana : s.n., 2012.

Vega, Irene Rodil Jiménez y Camino Pardo de. 2011. *Operaciones auxiliares con tecnologías de la información y comunicación.* España : Ediciones Paraninfo, S.A España, 2011.

Wake, William C. 2000. *Extreme Programming Explored.* 2000.

Zator Systems. 1990. Zator Systems. [En línea] 1990. www.zator.com/Hardware/H10_3.htm.