Universidad de las Ciencias Informáticas

Facultad 5

"Arquitectura de Plug-in para Sistemas de Visualización Médica"

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

Autora: Cecilia Valdespino Tamayo

Tutor(es): M.Sc. Osvaldo Pereira Barzaga

Ing. Yadira Ramírez Rodríguez

Co-Tutor: Ing. Ernesto Carrasco de la Torre

Ciudad de la Habana, Mayo 2011

"...aquí está una de las tareas de la juventud: empujar, dirigir con el ejemplo la producción del hombre de mañana. Y en esta producción, en esta dirección, está comprendida la producción de sí mismos..."

DECLARACIÓN DE AUTORÍA

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

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

_____ _____

Autora: Cecilia Valdespino Tamayo **Tutor:** M.Sc. Osvaldo Pereira Barzaga

_____ _____

Tutora: Ing. Yadira Ramírez Rodríguez **Co-Tutor:** Ing. Ernesto Carrasco de la Torre

DATOS DE CONTACTO

Generales del Tutor

Nombre y apellidos: M.Sc.Osvaldo Pereira Barzaga

Especialidad: Máster en Ciencias

Años de experiencias: 3 años

Correo electrónico: opereira@uci.cu

Teléfono de contacto: 837 2199

Generales de la Tutora

Nombre y apellidos: Ing. Yadira Ramírez Rodríguez

Especialidad: Ingeniería en Ciencias Informáticas

Años de experiencias: 4 años

Correo electrónico: yramirezr@uci.cu

Teléfono particular: 837 2703

Generales del Co-Tutor

Nombre y apellidos: Ing. Ernesto Carrasco de Torre

Especialidad: Ingeniería en Ciencias Informáticas

Años de experiencias: 1 año

Correo electrónico: ecarrasco@uci.cu

Teléfono de contacto: 837 2261

AGRADECIMIENTOS

A mis padres por haber estado siempre conmigo a pesar de todo, por apoyarme tanto en la vida personal como en la profesional, por sus consejos, por haberme guiado tanto y demostrarme que la vida es de sacrificio, porque significan lo más importante en mi vida y son el motor impulsor de cada paso que he decido dar.

A mis tutores Yadira, Osvaldo y Ernesto porque en este tiempo se convirtieron en mi amuleto, por dedicarme tanto tiempo a pesar de ser personas muy ocupadas, muchas gracias de todo corazón porque de cada uno aprendí algo diferente y nunca lo voy a olvidar.

A las personas que estuvieron ahí a pesar de todo como Saily y Dayana y Yeni, les agradezco mucho porque cada vez que las necesité para mi tesis estuvieron sin poner peros eso significó mucho para mí.

A mis compañeras con las cuales compartí todos estos años y no olvidaré y que de una forma u otra me apoyaron en toda mi vida universitaria: Ladi, Anna y Mailén.

A Daniel por haberme apoyado a pesar de la lejanía y por estar siempre al pendiente de todas mis cosas, muchas gracias mi amor.

A mis familiares y compañeros que estuvieron pendiente de mi vida como universitaria.

A Fidel, a la Revolución y a la Universidad por haberme dado la oportunidad de haber estudiado aquí y convertirme en una profesional.

DEDICATORIA

Le dedico mi tesis a:

Mi mamá, por ser la mujer más excepcional del mundo, porque para mí a pesar de todo serás la mejor madre que cualquier hijo(a) pueda tener, te quiero mucho.

Mi papá, porque me encaminó siempre en mis estudios, porque me inculcaste el hábito de aprender y éste es el resultado, no puedo tener mejores padres de corazón igual te quiero mucho.

RESUMEN

El presente trabajo pretende conformar la propuesta de una arquitectura de software robusta y confiable que permita a los Sistemas de Visualización Médica ser una aplicación extensible, personalizada y reusable.

Se realizó el análisis de los elementos relacionados con la Arquitectura de Software, los principales conceptos, estilos y patrones arquitectónicos y algunos sistemas que tienen integrada una arquitectura de plug-in.

Se describió la propuesta arquitectónica para los Sistemas de Visualización Médica usando el lenguaje de programación C++. La descripción de la arquitectura se basó en la Metodología de Desarrollo de Software Proceso Unificado de Desarrollo (RUP) usando como herramienta CASE Visual Paradigm.

Se propuso y se ejecutó una estrategia de evaluación de arquitectura de software con el objetivo de identificar los riesgos y fortalezas de la propuesta y poder tomar medidas correctivas a tiempo.

Como resultado se obtuvo una Arquitectura de plug-in sobre el framework Qt que permite un alto nivel de funcionamiento a la aplicación desarrollada, siendo así un sistema extensible y reusable, cubriendo las necesidades existentes en el proyecto Vismedic de la Facultad 5.

Palabras claves: Arquitectura de Software, Plug-in, Visualización Médica.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TICs) se han convertido en el vehículo común en el tránsito de la vida social de todos, Influyendo de forma favorable en el desarrollo de cada una de las áreas que rigen la sociedad. Dentro de estas se encuentra la medicina la cual no ha quedado exenta de estos adelantos gracias a la implementación y modernización de la tecnología como por ejemplo: respiradores artificiales, los chips, la endoscopia y la construcción de algún software. Una de las ramas de la medicina que más se ha beneficiado con estos adelantos es la cirugía, posibilitando la realización de todo tipo de intervenciones quirúrgicas con el menor grado de invasión y molestia por parte del paciente.

En la actualidad la tecnología en las cirugías mínimamente invasivas está cambiando radicalmente en la manera que los médicos efectúan sus procedimientos quirúrgicos. En Cuba existe una extensa red de instituciones de salud que compone la organización en todo el país, contando con una atención médica de alto nivel científico y tecnológico en un ambiente confortable; destacándose el Centro Nacional de Cirugía de Mínimo Acceso, situado en el municipio 10 de Octubre de Ciudad de la Habana. Este centro lleva a efecto una política científica y planificada de generalización de la Cirugía de Mínimo Acceso a todo el país.

Para que el desarrollo de las intervenciones de mínimo acceso posean un alto nivel científico se requiere de utilización de sistemas de software donde a partir de una visualización tridimensional del órgano del paciente, los médicos especialistas puedan realizar un diagnóstico de patologías y planificación del acto quirúrgico; estos sistemas poseen un alto costo en el mercado internacional, producto a la situación anterior y al intenso bloqueo económico al que está sometida la nación, son imposibles de adquirir.

Con el objetivo de seguir incrementando el avance tecnológico en la medicina del país, en la Facultad 5 de la Universidad de las Ciencias Informáticas surge un proyecto llamado Vismedic, para el desarrollo de un Sistema de Visualización Médica tridimensional de apoyo al diagnóstico y planeación quirúrgica a partir de imágenes médicas digitales.

El sistema está desarrollado desde una perspectiva general con el objetivo de cubrir los requerimientos básicos de todos los Sistemas de Visualización Médica tridimensional como cargar, pre-procesar, segmentar y visualizar imágenes médicas; independientemente de ello los especialista médicos presentan exigencias cada vez mayores, donde si el médico desea que la aplicación cumpla con los requerimientos

de sus especialidades los desarrolladores tienen que modificar el código previamente programado. Además el sistema no permite el desarrollo por terceras partes (ampliamente utilizado en el mundo para extender aplicaciones informáticas); el usuario final no puede configurar la interfaz gráfica de la aplicación, obligándolo a tener activo todos los módulos que brinda; y las funcionalidades implementadas son altamente dependientes lo que trae consigo altos costos en tiempo y recursos, poca reusabilidad de las funcionalidades para el desarrollo de futuras aplicaciones especializadas en cualquier rama de la medicina o que tengan el mismo fin de Vismedic.

Por lo que el **problema científico** queda formulado de la siguiente manera:

¿Cómo aumentar la extensibilidad, reusabilidad y personalización de los Sistemas de Visualización Médica?

El **objeto de estudio** lo constituye la Arquitectura de Software.

Y el **campo de acción** la Arquitectura de Plug-in para los Sistemas de Visualización Médica.

Como **idea a defender** se tiene: La arquitectura de Plug-in definida para los Sistemas de Visualización Médica tributará a lograr un sistema reusable, extensible y configurable por parte de los usuarios finales.

Para ello se ha definido el siguiente **objetivo general**: Definir una arquitectura de plug-in para los Sistemas de Visualización Médica.

Para darle cumplimiento al objetivo anteriormente expuesto se plantearon las siguientes **tareas de la investigación:**

- Estudio de los principales conceptos que desenlazan el marco de la investigación para desarrollar el diseño teórico de la investigación.
- Caracterización de soluciones existentes para comparar con la solución propuesta.
- Selección de los patrones y estilos existentes para incorporarlos a la estructura de la arquitectura propuesta.

- Diseño de las 4+1 vistas arquitectónicas para establecer la línea base de la solución propuesta.
- Evaluación de la Arquitectura de Software propuesta, mediante métodos de evaluación para analizar e identificar riesgos potenciales en su estructura y sus propiedades.
- Validación de la Arquitectura definida para confirmar la solidez de la solución propuesta.

Con el objetivo de dar cumplimiento a las tareas antes expuestas, se hizo necesario el uso de diferentes **métodos de investigación**:

Métodos Teóricos:

- **"Histórico - Lógico":** Para determinar las tendencias actuales, en este caso se empleó para realizar el estudio de la trayectoria real de determinados elementos que servirán de guía para la construcción de una buena arquitectura de software.
- **"Analítico - Sintético":** Utilizado para procesar toda la información en particiones más pequeñas y fácil de comprender, y compactar esas partes que fueron analizadas formándolas más simples con ideas más claras y concisas.
- **"Modelación":** Este método se utilizó para esbozar la propuesta de solución mediante cada uno de los artefactos diseñados en la especificación de las vistas de la Arquitectura de Software.

Métodos Empíricos:

- **"Observación":** Mediante este método se realizó la investigación para conocer los detalles fundamentales sobre el desarrollo de la arquitectura de plug-in para los Sistemas de Visualización Médica a través de Vismedic.

Para una mejor comprensión del contenido expuesto en este documento estará organizado de la siguiente forma: Introducción, Tres Capítulos, Conclusiones, Recomendaciones, Bibliografía, Referencias Bibliográficas, Anexos y un Glosario de Términos.

En el **Capítulo 1: Fundamentación Teórica** se aborda detalladamente todo lo referente a la fundamentación teórica que sustenta este trabajo de diploma. Describe los conceptos básicos de Arquitectura de Software. Caracteriza los patrones y estilos arquitectónicos y además se explica brevemente el funcionamiento de la arquitectura de plug-in en soluciones existentes.

En el **Capítulo 2: Propuesta de Solución** se establece la línea base de la arquitectura propuesta a través de las vistas arquitectónicas que propone la metodología de desarrollo de software RUP: Vista de casos de uso, Vista lógica, Vista de implementación, Vista de despliegue y Vista de procesos.

En el **Capítulo 3: Validación de la Propuesta de Solución** se evalúa la arquitectura previamente definida a través de los métodos de evaluación existentes para verificar que la propuesta realizada cumple con las especificaciones realizadas en el Capítulo 2 y se identifican sus riesgos potenciales.

1

CAPÍTULO
FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se realiza el estudio sobre el marco teórico referente a los temas: Plug-in, Arquitectura de Software, Estilos Arquitectónicos y Patrones, donde a partir de éstos se seleccionan los indicados según sus características para el desarrollo de la investigación. Se describe, además, las características de la herramienta case y metodología de desarrollo de software a utilizar. De la misma forma se realiza un análisis del funcionamiento de la arquitectura de plug-in en algunas soluciones existentes.

1.1 Definición de Arquitectura de Software

La Arquitectura de Software es una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones (2).

Es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución (1).

A partir del análisis de los conceptos antes mencionados se puede concluir: *que esta disciplina es el pilar fundamental en el desarrollo de software y es precisamente su concepción lo que permite establecer una línea común de trabajo a los integrantes de un proyecto para alcanzar los objetivos de un sistema de información, cubriendo todas las necesidades del mismo a través de la definición de un conjunto de patrones y abstracciones coherentes para su desarrollo.*

La arquitectura de software es de especial importancia ya que, la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de éste para satisfacer lo que se conoce como los atributos de calidad del sistema. Provee un conjunto de beneficios para un correcto desarrollo:

- Además de los atributos de calidad, la arquitectura de software juega un papel fundamental para guiar el desarrollo.
- Una de las múltiples estructuras que la componen, se enfoca en dividir el sistema en componentes que serán desarrollados por individuos o grupos de individuos.
- La identificación de esta estructura de asignación de trabajo es esencial para apoyar las tareas de planeación del proyecto.
- Los diseños arquitectónicos que se crean en una organización pueden ser reutilizados para crear sistemas distintos.
- Permite reducir costos y aumentar la calidad, sobre todo si dichos diseños han resultado previamente exitosos en otros sistemas.

1.2 Modelos de la Arquitectura de Software

Los modelos de arquitectura de software establecen las pautas básicas para definir una arquitectura de software para el desarrollo de aplicaciones en dependencia de la lógica del negocio que manipulen.

A continuación se especifican modelos que pueden responder al desarrollo de la propuesta de solución.

1.2.1 Arquitectura como etapa de ingeniería y diseño orientado a objetos

La arquitectura como etapa de ingeniería y diseño orientado a objetos es un modelo creado por James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, el cual está ligado al lenguaje de modelado UML y a la metodología de desarrollo de software RUP (Proceso Unificado de Rational).

Este modelo posee una serie de características que permiten una mayor comprensión del mismo, entre las que se destacan:

- Tiene predilección por un modelado denso y una profusión de diagramas.
- No utiliza ADLs (Lenguajes de descripción de Arquitectura de Software), reconocidos y utilizados por la academia de la AS (Arquitectura de Software).
- Utiliza un lenguaje unificado de modelado (por sus siglas en inglés UML) para las descripciones arquitectónicas.
- Concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico, susceptibles de ser descriptas a través de las vistas clásicas del modelo 4+1 de Kruchten (5).

1.2.2 Arquitectura basada en patrones

Esta tendencia reconoce que la importancia de un modelo emanado del diseño OO (Orientada a objetos), representa un pilar fundamental en la construcción de la arquitectura en cuestión. Pues en esta manifestación de la AS prevalece cierta tolerancia hacia modelos de procesos tácticos, no tan macroscópicos. El diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura. Es una corriente que no se encuentra tan rígidamente vinculada a UML (5).

1.2.3 Arquitectura basada en escenarios

Es Dentro de las arquitecturas existentes la basada en escenarios es la tendencia más novedosa. Se trata de un movimiento predominantemente europeo, con centro en Holanda. Recupera el nexo de la AS con los requerimientos y la funcionalidad del sistema, ocasionalmente borroso en la arquitectura estructural clásica. El movimiento se caracteriza por:

- Es una especialización de la AS procesual.
- Suele utilizarse diagramas de casos de uso, UML como herramienta informal u ocasional.
- Los casos de uso no están orientados a objeto.

- Los autores vinculados con esta modalidad han sido, en su mayoría, los codificadores de algunos métodos de evaluación de arquitectura del Instituto de Ingeniería de Software (por sus siglas en ingles SEI) (6).

1.3 Patrón

Un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio (2).

Cada patrón describe un problema que ocurre una y otra vez en el ambiente, y luego describe el núcleo de la solución de este problema, de tal manera que puede usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces.

Entre sus características se puede apreciar:

- Los patrones ayudan a construir la experiencia colectiva de Ingeniería de Software.
- Son una abstracción de “problema-solución” y se ocupan de problemas recurrentes.
- Identifican y especifican abstracciones de niveles más altos de componentes o clases individuales; y también proporcionan un vocabulario y entendimiento común (6).

1.3.1 Clasificación de los patrones

Los patrones se clasifican según el tipo de solución que brindan a una problemática determinada; a continuación se detallan las diferentes catalogaciones que existen en el desarrollo de un software:

- **Patrones de Análisis:** Usualmente específicos de aplicación o industria. Resuelve problemas de modelado del dominio, completitud, integración, equilibrio de objetivos múltiples y de planeamiento para capacidades adicionales comunes (1).
- **Patrones de Proceso o de Organización:** Tratan todo lo relacionado con el desarrollo o los procesos de administración de proyectos, técnicas o estructuras de organización, resolviendo problemas de productividad, comunicación, efectividad y eficiencia (1).

- **Patrones de Idioma:** Son estándares de codificación y proyecto, creados para resolver las operaciones comunes bien conocidas en un nuevo ambiente o a través de un grupo, la legibilidad y la predictibilidad.

- **Patrones de Arquitectura:** Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, rendimiento, modularidad y acoplamiento (1).

- **Patrones de Diseño:** Resuelven problemas de multiplicación de clases y adaptabilidad a requerimientos cambiantes, proponiendo como solución: Comportamiento de factoría, Clase-Responsabilidad - Contrato (CRC). En esta clasificación se encuentran los patrones GRAPS (Patrones de diseño para asignar responsabilidades) que describen los principios fundamentales de la asignación de responsabilidades a objetos y los GoF (Banda de los cuatro) que enmarcan los llamados patrones de diseño Estructurales, Creacionales y de Comportamiento (1).

1.4 Estilos arquitectónicos

Los estilos arquitectónicos definen las reglas generales de organización en términos de un patrón y las restricciones en la forma y la estructura de un grupo numeroso y variado de sistemas de software (1).

Se aplican a un nivel muy alto de abstracción en el cual no interesa saber cuál es la semántica de los elementos que se utilice, sólo se habla de filtros, tubos u objetos sin interesar lo que están representando.

Entre las razones que justifican que los estilos arquitectónicos constituyen un aspecto importante en el desarrollo de una arquitectura de software se pueden evidenciar:

- Sirven para sintetizar estructuras de soluciones.

- Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas.

- Definen los patrones posibles de las aplicaciones.

- Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

Los principales estilos arquitectónicos se encuentran divididos en la actualidad por clases de estilos que engloban una serie de patrones arquitectónicos.

A continuación se especifican dichas clasificaciones:

- Estilos de Flujo de Datos
    - Tubería y filtros
    - Estilos Centrados en Datos
    - Arquitectura de Pizarra o Repositorio
- Estilos de Llamada y Retorno
    - Modelo Vista Controlador (MVC)
    - Arquitecturas en Capas
    - Arquitecturas Orientadas a Objetos
    - Arquitecturas Basadas en Componentes
- Estilos de Código Móvil
    - Arquitectura de Máquinas Virtuales
- Estilos Heterogéneos
    - Sistemas Control de Procesos
    - Arquitecturas Basadas en Atributos
- Estilos Peer-to-Peer
    - Arquitecturas Basadas en Eventos

- Arquitecturas Orientadas a Servicios
- Arquitecturas Basadas en Recursos

- Plug-in (7)

1.4.1 Estilo arquitectónico Plug-in

Una plug-in (extensión) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúa por medio de la Interfaz Programada de la Aplicación (por sus siglas en inglés API) (8).

A continuación se muestra la estructura que propone dicho estilo:

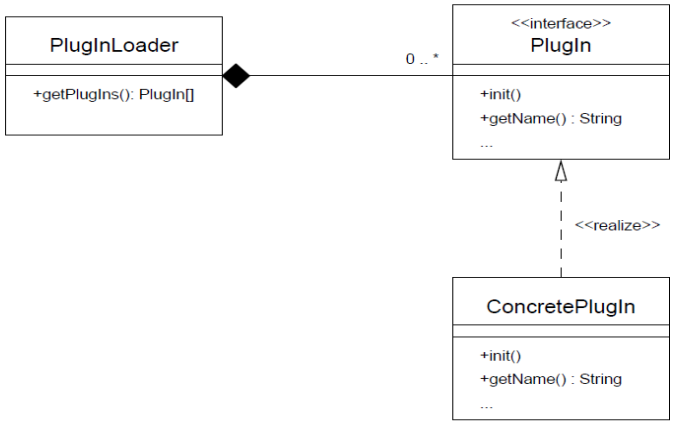

Figura. 1 Esquema del patrón plug-in

Dicho estilo no contiene ninguna variante arquitectónica, se ve de forma independiente. Se enfoca en explicar cómo se puede diseñar una aplicación con el fin de soportar varios plug-in permitiendo que la misma se extienda en tiempo de ejecución mediante la carga dinámica de módulos o clases que no conoce durante la compilación.

Su uso se basa en el cumplimiento de los siguientes requerimientos:

- Necesidad de expansión durante el tiempo de ejecución de eventos posiblemente desconocidos en el momento de inicialización.
- Modularización de sistemas muy grandes para reducir su complejidad.
- Desarrollo independiente de los componentes del sistema sin modificar otros módulos o reconstruir el mismo.
- Permitir el desarrollo de terceros contando sólo con en el conocimiento de la interfaz.
- Permitir el fácil desarrollo de nuevas características y actualizaciones, después de inicializada la aplicación.
- Disminuir el tiempo de arranque de la aplicación y requerimientos de hardware, especialmente de memoria (carga funcionalidades según sea necesario).
- Crear flexibilidad en los servidores que están en ejecución por mucho tiempo que no pueden ser reiniciados.

1.5 Tecnologías y herramientas utilizadas

Dentro de las herramientas utilizadas para establecer la línea base de la arquitectura de los Sistemas de Visualización Médica se encuentran: la metodología de desarrollo de software RUP con su lenguaje de modelado UML y el uso de la herramienta CASE Visual Paradigm para modelar cada uno de los artefactos arquitectónicamente significados que serán generados en la propuesta de solución.

A continuación se detallen sus principales elementos:

La metodología de desarrollo de software ***RUP (Proceso Unificado de Rational, Rational Unified Process en inglés)*** tiene como objetivo entregar un producto de software. Constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Posee una forma disciplinada de asignar tareas y responsabilidades. Sus características fundamentales se basan en un desarrollo iterativo e incremental, guiado por casos de uso y centrado en la arquitectura. Utiliza a UML para especificar, visualizar y documentar los artefactos que se crean durante el proceso de desarrollo. Su ciclo de vida está enmarcado en cuatro fases de desarrollo: Inicio,

Elaboración, Construcción y Transición por las cuales circulan nueve disciplinas divididas en seis de ingeniería: Modelamiento del negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba y Despliegue, y tres de apoyo: Gestión de Proyecto, Gestión de la Configuración y el Cambio y Ambiente.

Esta metodología propone una guía para establecer la línea base de la arquitectura de un proyecto de desarrollo de software basada en la modelación de las 4 + 1 vistas.

Visual Paradigm constituye la herramienta case que lleva a cabo la modelación de los artefactos que forman parte de la propuesta de solución. Herramienta profesional multiplataforma que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite modelar los diferentes diagramas generados en el desarrollo de un software, así como generar código desde diagramas, entre otros elementos que pueden ser de ayuda en la documentación de un sistema determinado. Posee una buena integración con Entornos multi-plataforma de Desarrollo Integrado (IDEs), incluye localización en castellano, muy personalizable y soporta varios lenguajes de programación.

1.6 Pasos para definir la Arquitectura de Software

El establecimiento de una arquitectura de software se apoya en el desarrollo de los pasos que a continuación se detallan:

- Definir la infraestructura de desarrollo, dígase identificar los elementos con los cuales se cuenta: computadoras, dispositivos, sistema operativo, redes de comunicación y servidores, entre otros e identificar las propiedades del sistema globalmente como el tipo de aplicación que se está construyendo, si es una aplicación interactiva, centralizada, distribuida, etc.

- Seleccionar los patrones arquitectónicos con el tipo de aplicación que se está construyendo y con la infraestructura que se cuenta y luego combinarlos

- Identificar subsistemas funcionales utilizando los métodos de análisis y diseño que se conocen, llenando la distribución que se decidió desarrollar y luego integrar los subsistemas funcionales con la infraestructura que va a tener el sistema.

La arquitectura es un extracto de los modelos que se encuentran en la línea base la arquitectura, RUP propone para diseñar cada uno de sus artefactos tener en cuenta las 4+1 vistas. A continuación se detallan los elementos arquitectónicamente significativos presentes en cada una de ellas:

- ***Vista de Casos de Uso:*** Representa un subconjunto del artefacto del modelo de casos de uso y lista los casos de usos o escenarios del modelo de caso de uso más significativos, con las funcionalidades centrales del sistema. Si éste se hace extenso entonces deberían ser organizadas por paquetes lo cual facilitaría la compresión de esta vista. Así, desde casos de uso se debiera poder hacer una trazabilidad a todos los componentes del sistema de software, viendo por ejemplo, qué máquinas, clases, componentes o procesos son los responsables de que el sistema cumpla una cierta funcionalidad.

- ***Vista Lógica*****:** Representa un subconjunto de artefactos del modelo del diseño la cual representa los elementos del diseño más importantes para la arquitectura del sistema. Esta vista describe las clases más importantes, su organización en paquetes y subsistemas y realizaciones de casos de uso más significativos como por ejemplo las relaciones que describen aspectos dinámicos del sistema. La notación más usada es UML (Lenguaje Unificado de Modelado) y dentro de ésta la utilización de los diagramas de clases y paquetes.

- ***Vista de Procesos:*** Suministra una base para la comprensión y organización de los procesos de un sistema ilustrado por medio del mapeo de clases y subsistemas en procesos e hilos, como suele usarse cuando el subsistema presenta procesos concurrentes. Estos procesos que se representan son los más importantes y las clases que intervienen en los mismos. La notación más usada es UML, y dentro de ésta la utilización de los diagramas de estados, actividad y similares. Se pueden encajar varios estilos, por ejemplo pueden usarse tuberías y filtros (pipes and filtres) o Cliente – Servidor (con variantes de múltiples clientes – simple servidor y múltiples clientes – múltiples servidores).

- ***Vista Implementación:*** Describe la descomposición del software en capas y subsistemas de implementación, también provee una vista de la trazabilidad de los elementos del diseño desde la vista lógica hasta la implementación. Vale destacar que podrá describirse la vista de

implementación, por completo, solamente después de haber identificado todos los elementos software. La notación más usada es UML, y dentro de ésta la utilización de los diagramas de componentes y paquetes.

- ***Vista de Despliegue:*** Suministra una base para la comprensión de la distribución física de un sistema a través de los nodos, puede utilizarse cuando dicho sistema está distribuido. Existe una traza directa del modelo de implementación hasta este modelo puesto que cada componente físico debe estar almacenado en un nodo lo que incluye la asignación de tareas provenientes de la vista de procesos. Su representación se hace a través del artefacto Diagrama de Despliegue.

1.7 Análisis de algunas soluciones existentes

En la actualidad existen sistemas que utilizan los plug-in con el fin de hacer más extensible y reusable las mismas, pero son muy pocas las que tienen definida una arquitectura de plug-in robusta que expliquen el funcionamiento de la misma.

A nivel internacional existen aplicaciones con una estructura de plug-in definida entre las que se encuentra Entorno de Desarrollo Integrado (IDE) Eclipse y el framework Qt.

Eclipse proporciona un núcleo de servicios para el control de un conjunto de herramientas de trabajo con el fin de apoyar las tareas de programación, envolviendo sus herramientas en componentes llamados plug-in de Eclipse. Un plug-in, en eclipse, es un componente que proporciona un determinado tipo de servicio en el contexto de su entorno de trabajo. El mecanismo básico de la extensibilidad de este entorno es que los plug-in puedan agregar nuevos elementos de procesamiento a los ya existentes y proporciona un conjunto de plug-in básicos para iniciar dicho proceso (9).

Esta arquitectura, además, gestiona la dependencia entre los plug-in, es decir, para poder activar algún componente es necesario que otro esté funcionando en el sistema.

El ***framework Qt*** proporciona dos Interfaces de Aplicación (APIs) para la creación de plug-in:

- Un alto nivel de la API para escribir extensiones para Qt en sí mismo: los controladores personalizados de base de datos, formatos de imagen, códecs de texto, estilos personalizados, etc.
- Un bajo nivel de API para ampliar las aplicaciones Qt.

La arquitectura de plug-in que utiliza Qt, persigue el mismo objetivo que la aplicación antes expuesta, ya que permite que el usuario pueda extender cualquiera de las clases bases con la posibilidad de definir una clase implementada por él mismo como un plug-in, siempre y cuando la implementación esté en un archivo .cpp; algunas requieren funciones adicionales que deban aplicarse.

También propone hacer extensible la aplicación a través de plug-in estáticos los cuales representan los que aparecen por defecto mientras que los dinámicos son los que pueden ser agregados por el usuario al framework. Si se genera la versión estática de Qt, ésta es la única opción para la inclusión de plug-in predefinido de Qt. El uso de plug-in estático hace que el despliegue esté menos propenso a errores, pero tiene el inconveniente de que ninguna funcionalidad de complementos se puede agregar sin una reparación completa y la redistribución de la solicitud. Cuando se compila la aplicación como una biblioteca estática brinda una serie de plug-in con estas características previamente definidas a través de la opción “Importar plug-in estáticos”.

Aunque estas aplicaciones poseen un sistema de plug-in bien estructurado presentan como desventaja fundamental que la utilización de una de estas arquitecturas obliga al uso del entorno de desarrollo que ellas proveen.

Otras compañías de software utilizan la tecnología Java o .NET las cuales pueden extender el núcleo de sus sistemas, pero los diseños arquitectónicos utilizados en el desarrollo de estos sistemas no permiten que los mismos sean multiplataforma; pues no se tiene acceso a la documentación que sustenta la arquitectura que guía su desarrollo.

En el estudio a nivel nacional se tienen los desarrollos de software implementados en la Universidad de las Ciencias Informáticas donde existen varios sistemas como el proyecto SCADA (Sistemas de Supervisión, Control y Adquisición de Datos) perteneciente al Centro de Desarrollo Industrial (CEDIN) el

cual está desarrollado sobre el IDE Eclipse permitiéndole ser un sistema extensible a través de la carga de módulos independientes en forma de plug-in; y el proyecto AlasPACS (por sus siglas en inglés Picture Arquiring and Comunication Systems) del Centro de Informática Médica (CESIM) que tiene una arquitectura de plug-in definida pero totalmente dependiente a .NET. Dichos sistemas no poseen una arquitectura genérica para su desarrollo.

1.8 Consideraciones parciales

A raíz del estudio realizado hasta el momento fue posible la toma de las siguientes decisiones:

- Se definió como modelo de arquitectura a tener en cuenta la Arquitectura como etapa de ingeniería y diseño orientado a objetos.
- Se selecciona como metodología de desarrollo de software RUP debido a que es centrada en la arquitectura, es aplicable a proyectos a largo plazo intentando reducir el costo y la realización del mismo y es capaz de adaptarse a las características y complejidad de cualquier proyecto de software.
- Se escoge Visual Paradigm como herramienta de modelado porque es multiplataforma, es considerado un producto de calidad y utiliza a UML como lenguaje de modelado.
- Se decidió diseñar una arquitectura de plug-in propia debido a que las soluciones existentes no son factibles para la solución propuesta.

2

CAPÍTULO
PROPUESTA DE SOLUCIÓN

La definición de una Arquitectura de Software aporta en particular una visión abstracta de alto nivel al realizarse el diseño; es por ello que en este capítulo se realiza la descripción de la propuesta de solución a partir de la definición de las 4+1 vistas, generando cada uno de sus artefactos así como las restricciones arquitectónicamente significativas que se debe tener en cada una de ellas.

2.1 Modelo de dominio

El modelo del dominio es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema.

El entorno en el cual está enmarcado el problema que desencadena esta investigación se desarrolla de la siguiente forma: *un médico le orienta a un paciente una serie de estudios médicos que generan un conjunto de imágenes, las cuales son generadas por las tomografías computarizadas (TAC) o resonancias magnéticas (MRI); donde a partir del análisis de las mismas el médico emite un diagnóstico.*

En la **Figura 1** ilustra la descripción del negocio anteriormente descrito a través de un modelo de dominio.

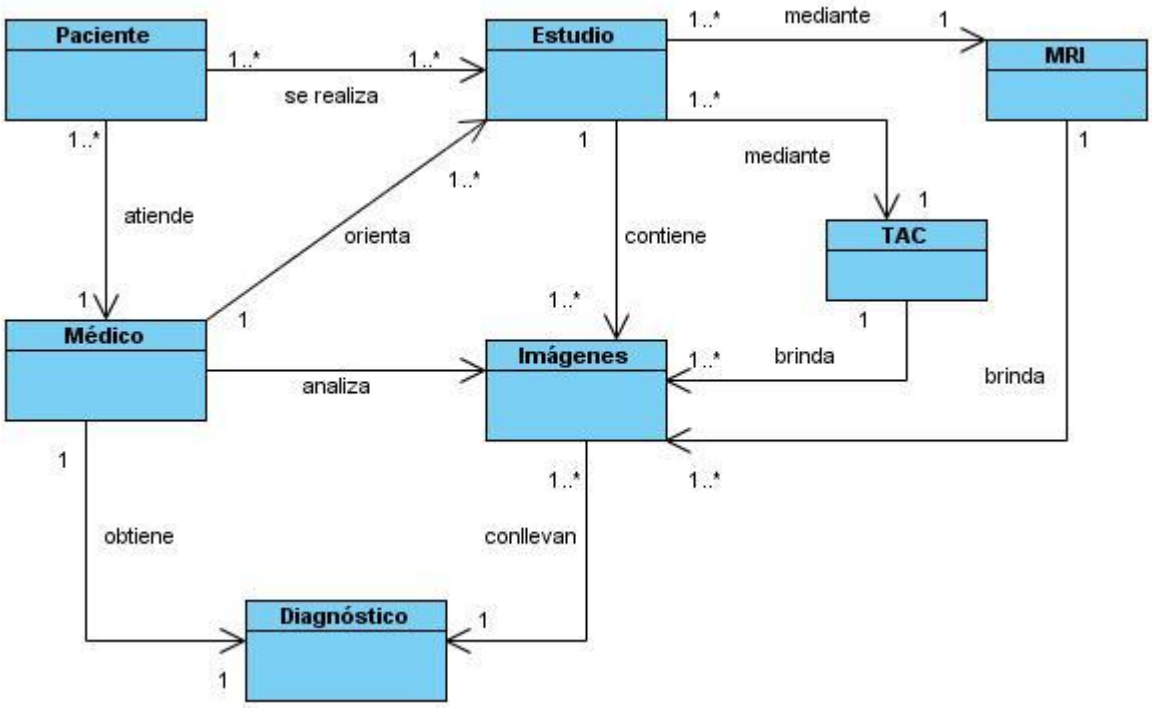

Figura. 2 Modelo del dominio

A continuación se describen los conceptos que forman parte del negocio ilustrado:

El ***Médico*** es la persona capacitado en el manejo y conocimiento de los métodos de Imagenología para emitir un diagnóstico.

Un ***Estudio*** es el examen orientado por el médico al paciente para obtener una patología y emitir un diagnóstico.

Las ***Imágenes*** constituyen el resultado del estudio orientado por el médico al paciente a través del TAC y MRI.

El ***Paciente*** es aquel que recibe los servicios de un médico u otro profesional de la salud, sometiéndose a un examen.

Una ***MRI*** (Resonancia Magnética) es una prueba que realiza una fotografía de los órganos y tejidos internos sin exponer al paciente a radiaciones. Utiliza un campo magnético poderoso y un tipo de frecuencia de radio que realiza imágenes computarizadas.

Una ***TAC*** (Tomografía Axial Computarizada) es un procedimiento de diagnóstico médico que utiliza rayos X con un sistema informático que procesa las imágenes y permite obtener imágenes radiográficas en secciones progresivas de la zona del organismos estudiada, y si es necesario, imágenes tridimensionales de los órganos o estructuras orgánicas.

El ***Diagnóstico*** es el resultado que da el médico luego de realizar un estudio a un paciente.

2.2 Requisitos funcionales del software

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, y se mantienen invariables sin importar con qué propiedades o cualidades se relacionen (8).

A partir de la utilización de la Técnica de Recopilación de Información, entrevista, se identificaron las funcionalidades deseadas por el cliente; las cuales se especifican a través de los siguientes requisitos funcionales:

No	**Funcionalidad**	**Descripción**	**Prioridad**
RF1	Importar plug-in.	El sistema debe permitirle al usuario seleccionar de un directorio el plug-in que desea añadir a la aplicación.	Alta
RF2	Desinstalar plug-in.	El sistema debe mostrar el listado de plug-in importados para que el usuario pueda seleccionar uno y	Alta

		desinstalarlo.	
RF3	Habilitar plug-in	El sistema debe permitir al usuario activar las funcionalidades de un plug-in que ha sido deshabilitado.	Alta
RF4	Deshabilitar plug-in.	El sistema debe permitir al usuario desactivar las funcionalidades de cualquier plug-in activo.	Alta

Tabla 1 Especificación de requisitos funcionales

2.3 Restricciones arquitectónicas

La definición de una arquitectura de software debe poseer un conjunto de reglas que se deben de seguir para el desarrollo de un software las cuales constituyen restricciones arquitectónicas.

Para el desarrollo de la propuesta de solución se tiene:

- Las propiedades de hardware dependen de los requerimientos de los sistemas que requieran utilizar la arquitectura propuesta.
- La arquitectura está diseñada en el lenguaje de desarrollo C++.
- El desempeño de la aplicación debe ser muy eficiente de tal manera que el usuario inmediato y todos los demás observen rápidamente los cambios realizados en un momento determinado.
- El diseño debe ser orientado por y para la comodidad del usuario, de manera que la interfaz sea intuitiva y fácil de manejar, al mismo tiempo que se fomente altamente la interacción entre ambos.

- Los plug-in deben tener los mínimos privilegios necesarios para correr. Estos privilegios varían según el tipo de aplicación que se esté haciendo.
- La solución propuesta se encontrará centralizada en una computadora en los centros salud especializados en la cirugía de mínimo acceso con las prestaciones expuestas anteriormente, donde los especialistas accederán a la aplicación mediante un protocolo de comunicación que permite la transferencia en tiempo real.

2.4 Vista de casos de uso

La vista de casos de uso representa el comportamiento del sistema tal y como lo percibe el usuario final, analistas y encargados de pruebas. A través de la misma se muestra un subconjunto del artefacto del modelo de casos de uso y la lista de los casos de uso o escenarios de los modelos más significativos con las funcionalidades centrales del sistema.

Para la realización de propuesta de solución se identificaron cuatro requisitos funcionales que conllevaron a cuatro casos de uso con categoría crítica, por lo que constituyen casos de uso arquitectónicamente significativos.

2.4.1 Casos de uso arquitectónicamente significativos

Los casos de uso arquitectónicamente significativos son aquellos que representan las partes más críticas del sistema, es decir, aquellos que cubren las principales funcionalidades del sistema y responden a las funcionalidades primordiales que desea el cliente.

La figura 3 ilustra el modelo de caso de uso que forma parte de la vista de casos de uso con los actores y casos de uso arquitectónicamente significativos.

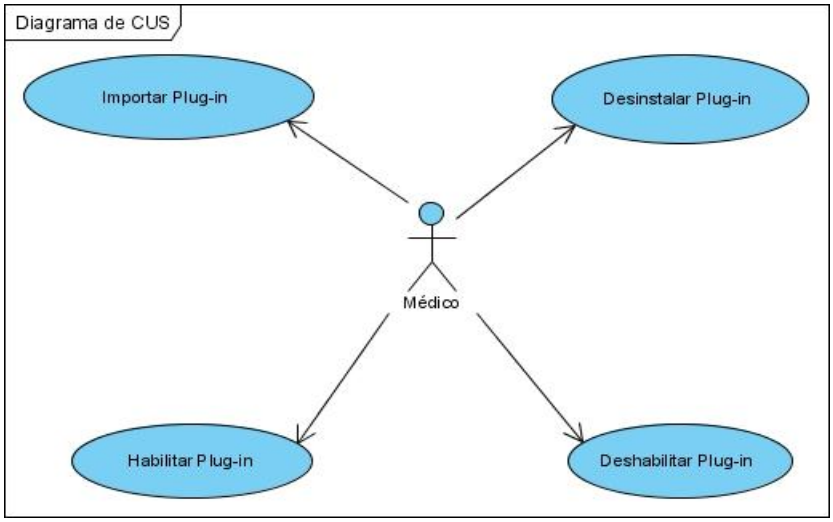

Figura. 3 Diagrama de Casos de Usos del Sistema

A continuación se detallan brevemente las funcionalidades básicas que forman parte de la propuesta de solución:

- ***Importar Plug-in***: El caso de uso inicia cuando el médico selecciona la opción importar, el sistema le muestra una ventana de búsqueda, se selecciona un plug-in y el sistema lo coloca en la lista de plug-in importados.

- ***Desinstalar Plug-in***: El caso de uso inicia cuando el médico accede a la interfaz Manager Plug-in, selecciona el plug-in que desea desinstalar, presiona el botón desinstalar y el sistema lo elimina de la lista de plug-in activos.

- ***Habilitar Plug-in***: E El caso de uso inicia cuando el médico accede a la interfaz Manager Plug-in, selecciona el plug-in que desea habilitar, presiona el botón habilitar y el sistema habilita el plug-in.

- ***Deshabilitar Plug-in***: El caso de uso inicia cuando el médico accede a la interfaz Manager Plug-in, selecciona el plug-in que desea deshabilitar, presiona el botón deshabilitar y el sistema lo deshabilita de la lista de plug-in activos.

2.4.2 Realizaciones de los casos de uso

Las realizaciones de los casos de uso se utilizan para representar el flujo en términos de objetos del análisis. A continuación se muestran detalladas en términos de objetos las descripciones de los casos de uso realizadas anteriormente mediante los diagramas de colaboración, los cuales ilustran la interacción entre objetos y el orden secuencial en el que ocurren dichas interacciones, es decir, cómo se comunican los objetos entre sí:

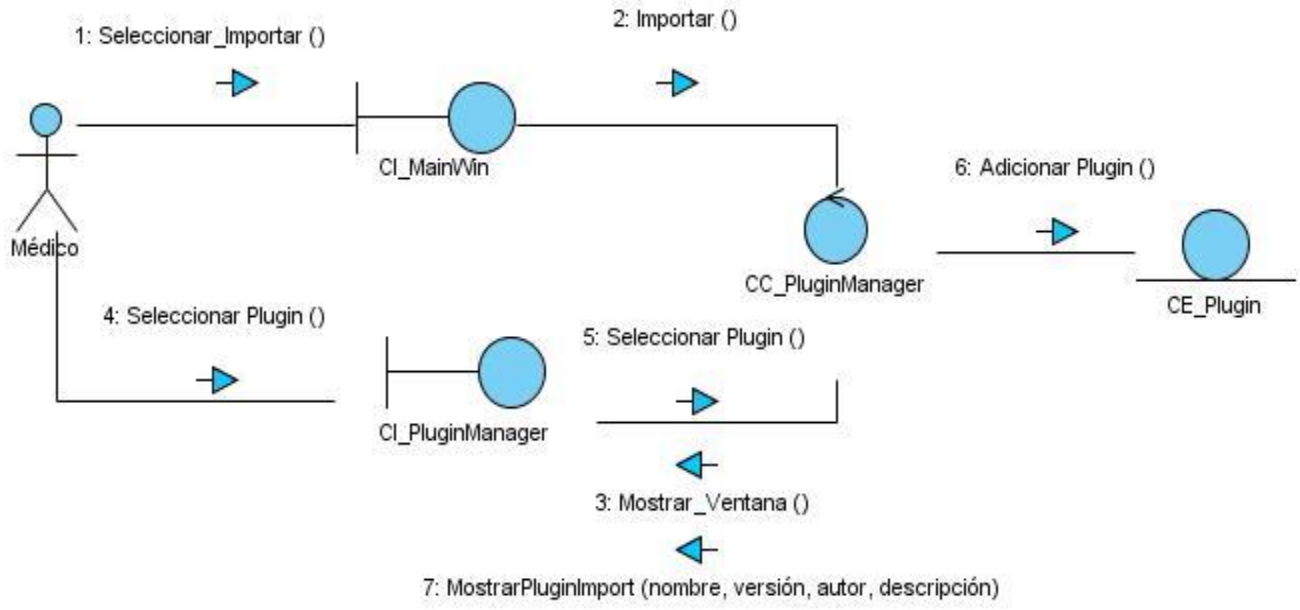

Figura. 4 Diagrama de colaboración CU Importar

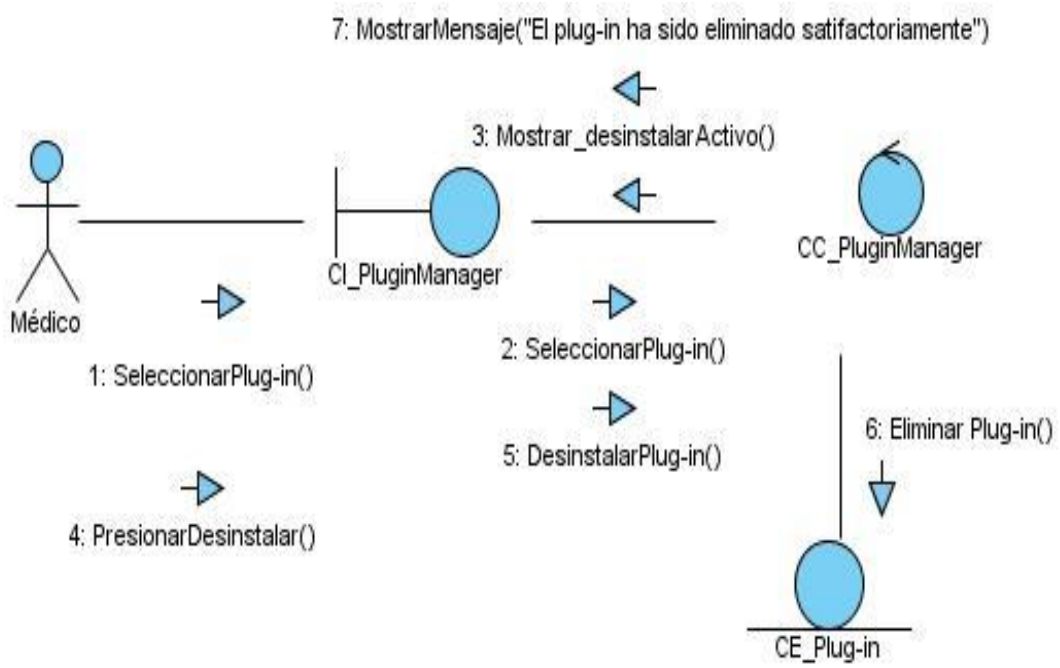

Figura. 5 Diagrama colaboración CU Desinstalar

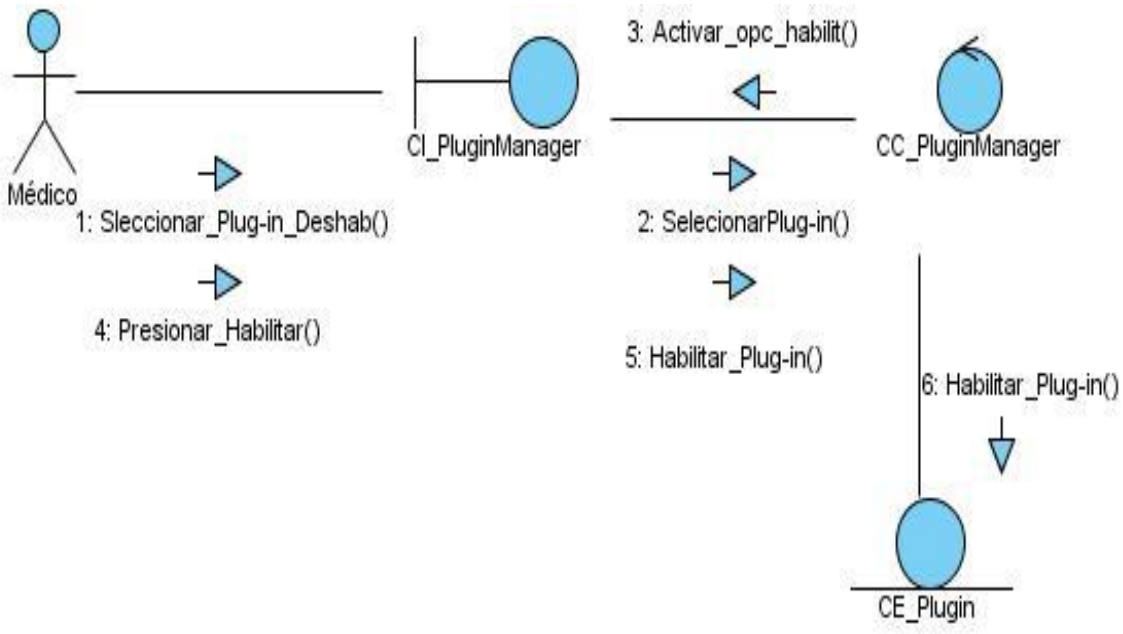

Figura. 6 Diagrama de colaboración CU Habilitar

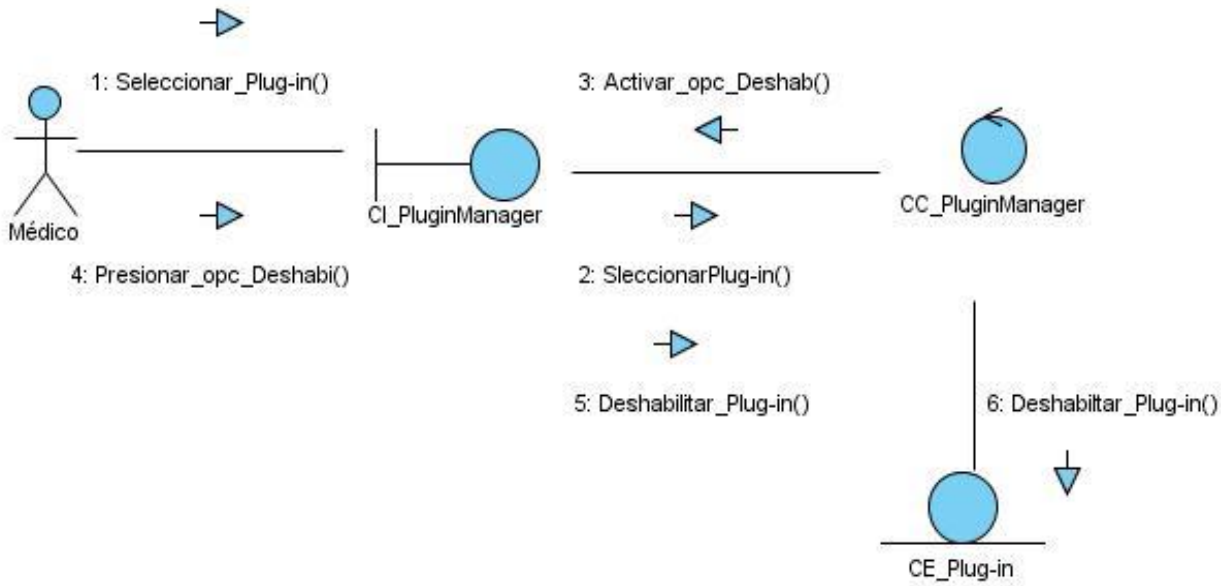

Figura. 7 Diagrama de colaboración CU Deshabilitar

2.5 Vista lógica

La vista lógica comprende los elementos del diseño significativos para la arquitectura. El proyecto se divide en paquetes de diseño para facilitar el desarrollo de los mismos, estos paquetes son independientes, de forma que si cambian los requerimientos de un paquete los otros no se verán afectados.

2.5.1 Descomposición en paquetes

Los paquetes del diseño constituyen una herramienta organizacional, utilizados en la solución para dividir el modelo de diseño en piezas más pequeñas y puedan ser de mejor entendimiento para el desarrollo de la arquitectura que se propone.

A continuación el diagrama por paquetes definido:

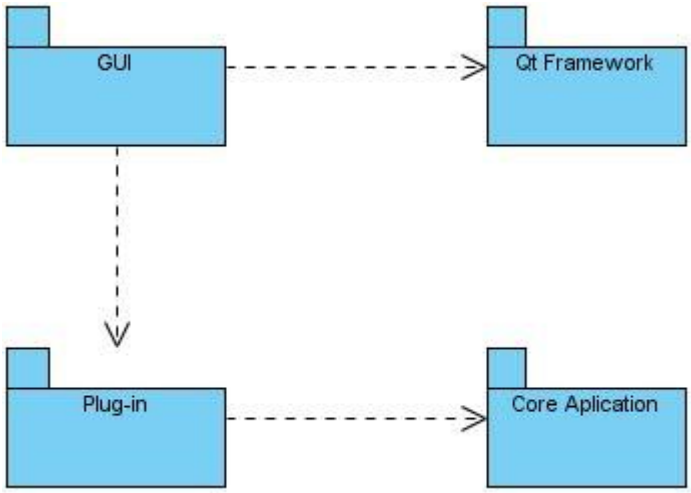

Figura. 8 Diagrama de paquetes

2.5.2 Descripción y diseño de los paquetes

A continuación se detallan los paquetes definidos en el diagrama de paquetes que forman parte de la propuesta de solución:

- ***GUI:*** Paquete encargado de mostrar la interfaces gráficas con la que va a interactuar el usuario a través de un conjunto de imágenes y objetos gráficos que permiten representar la información y las acciones disponible.

- ***Qt Framework:*** Paquete donde se encuentra el entorno de desarrollo sobre el cual debe estar diseñada la interfaz gráfica de la aplicación y servirá para agregar la interfaz que permitirá interactuar con los plug-in.

- ***Core Application:*** Paquete que contiene las clases que implementa la lógica que rige al Sistema de Visualización Médica con todas las técnicas y algoritmos de adquisición segmentación y procesamiento de imágenes.

- ***Plug-in:*** Paquete encargado de gestionar todas las operaciones que se deseen realizar sobre un plug-ni a partir de una jerarquía de clases, maneja todo lo relacionado con la carga, la activación,

desactivación y desinstalación de éstos. Dentro de este paquete se la estructura de clases que rigen las funcionalidades básicas definidas en la propuesta de solución. La figura 9 ilustra dicha estructura:

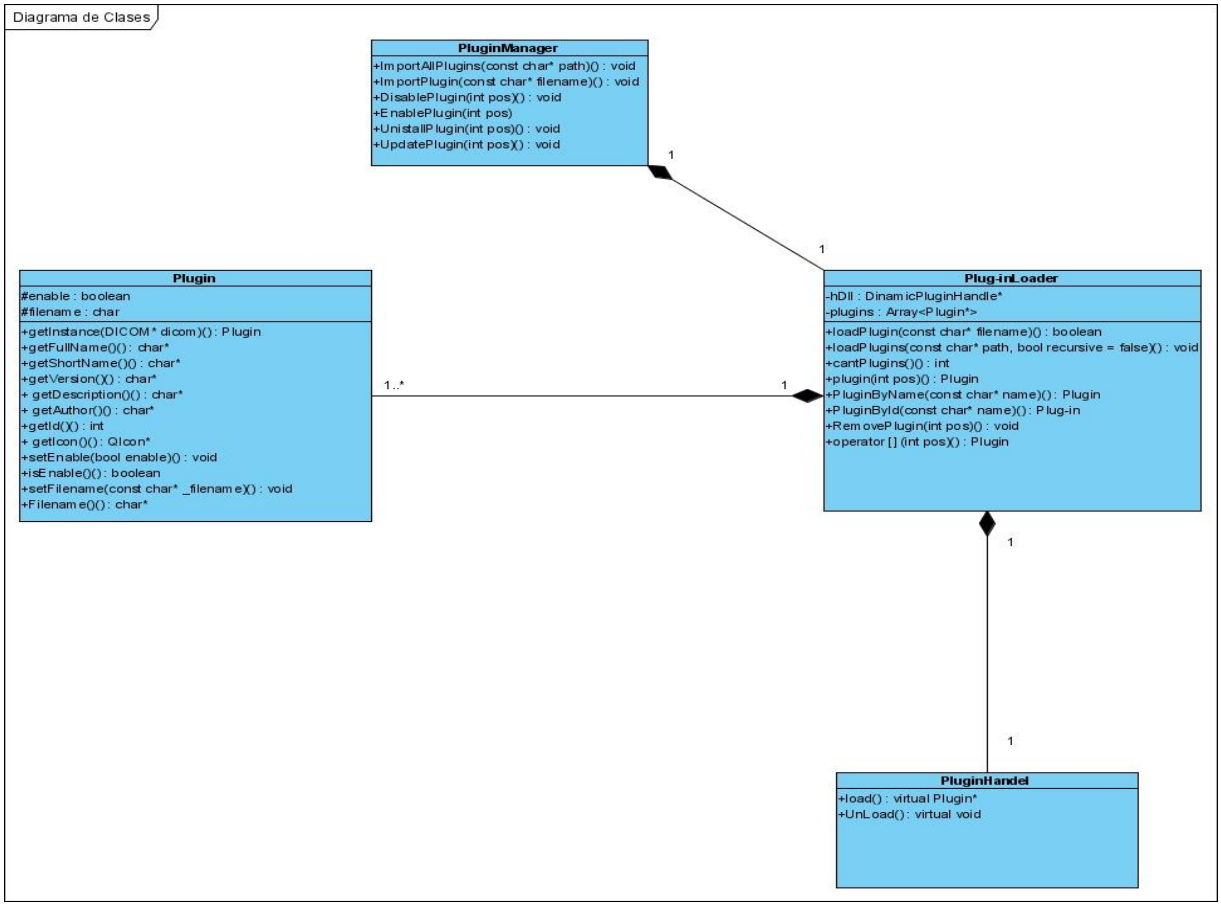

Figura. 9 Diagrama de clases

Breve descripción de las clases ilustradas en la figura 9:

- **Plugin:** Es una clase entidad que define la interfaz de un plug-in dentro de la arquitectura propuesta.

- **PluginHandel:** Es una clase interfaz que posibilita la carga de los ficheros en los cuales está almacenada la implementación del plug-in que se desea importar. En caso de estar en Windows esta clase gestionaría ficheros con extensión (*.dll) y en caso de Linux ficheros con extensión (*.so).
- **PluginLoader:** Es una clase que controla y gestiona el conjunto de objetos correspondientes a los plug-in importados por la aplicación.
- **PluginManager:** Es la clase donde se implementa toda la lógica del trabajo con los plug-in importados en la aplicación.

2.6 Vista de despliegue

La vista de despliegue muestra la configuración de los nodos (procesadores y dispositivos) que participan en la ejecución y de los componentes que residen en los mismos. Mostrando mediante los protocolos de comunicación como colabora un nodo con otro.

Para el caso de la arquitectura propuesta la distribución está en dependencia de la aplicación que necesite utilizar la misma. En la **Figura 10** se muestra como estará distribuido el sistema Vismedic sobre el cual se hará la validación de la solución que se propone.

Figura 10 Diagrama de Despliegue

A continuación se brinda una breve descripción de los nodos presentes en la figura anterior:

PC_Cliente: Representa la estación de trabajo que usará el cliente para interactuar con el Sistema de Visualización Médica.

Servidor: Representa la estación donde estarán almacenados todos los recursos del Sistema de Visualización Médica a la cual el cliente accederá para ejecutar las distintas funcionalidades que necesite.

RTP: Protocolo de comunicación para tiempo real que permite el envío de los números de secuencia, marcas de tiempo, origen y llegada de los datos, además de proveer información para la detección de paquetes perdidos.

2.7 Vista de implementación

La vista de implementación muestra los ejecutables y artefactos construidos para la implementación de la Arquitectura plug-in diseñada para la aplicación Vismedic.

A continuación se muestra la relación entre todos los componentes físicos que deben estar presentes en el desarrollo de la arquitectura y que responden a lo planteado anteriormente:

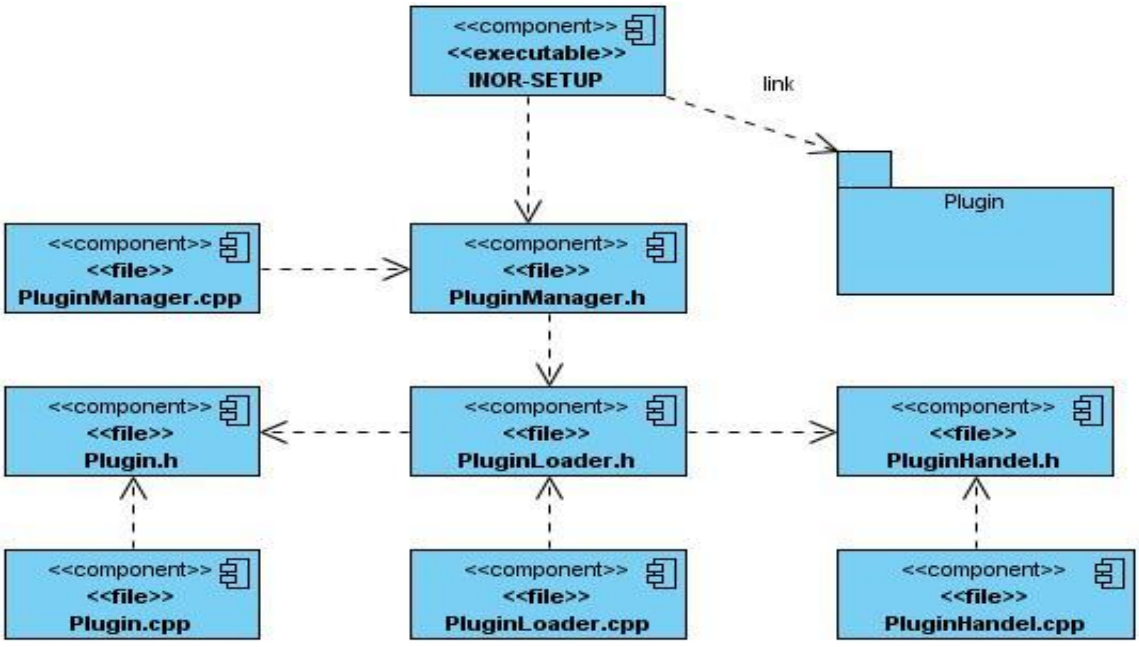

Figura 2 Diagrama de Componentes

El componente INOR-SETUP es el ejecutable que se utilizará para una mejor distribución de la aplicación a los clientes, el cual necesita de cada uno de los archivos que contienen el código fuente. El paquete

Plug-in contendrá todos los archivos *dll o *so (con el prototipo “library”) que tendrán también código fuente pero que será cargado por la aplicación en caso que sea necesario o si el propio usuario lo solicita, no se puede definir la cantidad de archivos que contendrá dicho paquete ya que representa un proceso dinámico que permitirá a la aplicación ser extensible y configurable.

2.8 Consideraciones parciales

En el presente capítulo se establecen:

- Las pautas arquitectónicamente significativas representadas a través de las 4+1 vista de la arquitectura: Vista de Casos de Uso, Vista Lógica, Vista de Despliegue y Vista de Implementación, las cuales dan lugar a la propuesta de solución: Arquitectura de Plug-in para Sistemas de Visualización Médica.

- La Vista de Procesos no se describe debido a que la línea base propuesta no conlleva en su desarrollo los elementos que ella requiere.

3

CAPÍTULO
VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

En el presente capítulo se abordará el cumplimiento de las restricciones expuestas anteriormente en la propuesta de solución mediante la evaluación de la arquitectura, lo cual constituye el medio de validación de la misma. Se hace alusión brevemente mediante su caracterización y pasos de aplicación de los métodos de evaluación que ofrecen mejores resultados de acuerdo a la solución que se ha propuesto, además de la descripción de los atributos de calidad que pueden ser evaluados a través de dichos métodos y por último se estima el comportamiento del sistema a desarrollar según los atributos de calidad identificados; los cuales constituyeron los puntos de partida para evidenciar los riesgos y no riesgos de la Arquitectura de Plug-in especificada.

3.1 Evaluación de la Arquitectura de Software

La evaluación de la arquitectura permite mitigar riesgos potenciales en el desarrollo del software. Desde etapas tempranas del proceso de desarrollo, el arquitecto de software tiene la posibilidad de ir refinando la arquitectura, al definir su línea base como se ha plasmado en el capítulo anterior, se responde al cumplimiento de los requerimientos funcionales básicos que debe poseer la aplicación, garantizando una primera iteración. Esta arquitectura puede ser refinada en las próximas iteraciones definidas en el ciclo de vida del sistema, donde será probada para evaluar el cumplimento de sus objetivos.

Una arquitectura robusta debe cumplir con las funcionalidades del sistema y con los atributos de calidad esperados del mismo. La parte ligada a estos atributos de calidad son los requisitos no funcionales del sistema, los que poseen gran relevancia ya que, especifican las características que se esperan del sistema: robusto, fiable, portable, usable, entre otros.

La arquitectura es el marco para todas las decisiones técnicas y tiene un importante impacto en el costo de los productos y la calidad. Una evaluación de la arquitectura no garantiza la alta calidad o el bajo costo de desarrollar los productos, pero puede señalar áreas de riesgo que, de ser tratadas en tiempo y con la calidad requerida, garantizaran una mayor calidad del producto final y disminución de su costo de desarrollo.

3.2 Atributos de calidad

Los atributos de calidad son requerimientos del sistema que hacen referencia a características que éste debe satisfacer.

Estos se clasifican en dos grandes grupos, los cuales son:

- **Observables vía ejecución o externos:** Aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución (9).
- **No observables vía ejecución o internos:** Aquellos atributos que se establecen durante el desarrollo del sistema (9).

Atributos de Calidad	Descripción
Disponibilidad	Es la medida en que el sistema se encuentra a disposición de quienes deben acceder a ella.
Confidencialidad	Es la ausencia de acceso no autorizado a la información.
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo

	del tiempo.
Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

Tabla 2. Atributos de Calidad Observables vía ejecución (9)

Atributos de Calidad	Descripción
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad	Es la ausencia de alteraciones inapropiadas de la información.
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad.
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de

	computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de Prueba	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

Tabla 3. Atributos de Calidad no observables vía ejecución (9)

3.3 Técnicas de evaluación

Las técnicas de evaluación de la arquitectura se clasifican en: cualitativas (escenarios, cuestionarios y listas de chequeo) y cuantitativas (métricas, normas, máximos y mínimos teóricos, simulaciones, prototipos, entre otros.), lo cual se ilustra en la figura 12. Estos tipos de técnicas posibilitan evaluar una arquitectura y establecer comparaciones entre arquitecturas candidatas para determinar cuál satisface más a un atributo de calidad específico.

Las técnicas de evaluación cualitativas son usadas cuando la arquitectura se encuentra en construcción y arrojan como resultados respuestas de sí o no. Estas técnicas posibilitan evaluar una arquitectura o hacer comparaciones entre arquitecturas candidatas y poder determinar cuál satisface más un atributo de calidad específico.

Las técnicas de evaluación cuantitativas se utilizan cuando la arquitectura ya ha sido implantada en el desarrollo de un sistema determinado.

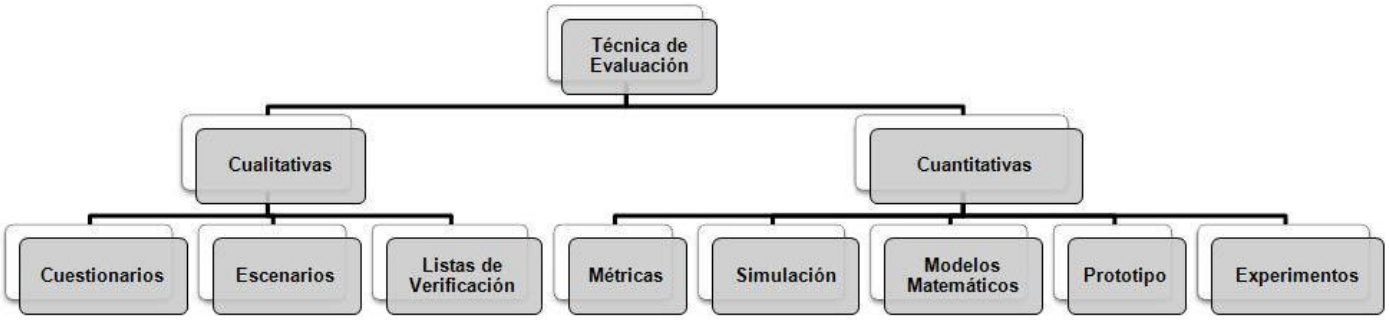

Figura 3. Clasificación de las técnicas de Evaluación

3.4 Métodos de evaluación de la arquitectura

Los métodos de evaluación de una arquitectura describen los pasos a seguir para evaluar una arquitectura de software. Existen múltiples métodos para realizar pruebas a la arquitectura de software, cada uno con sus características específicas; éstos deben ser desarrollados por roles o implicados capacitados para evaluar en cuanto a atributos de calidad la arquitectura desarrollada.

Para la utilización de estos métodos es necesario tener claro las características fundamentales que debe de poseer el software, para poder escoger el método ideal de acuerdo a dichas características, lo cual permitirá arrojar las fortalezas y debilidades del sistema evaluado.

Entre los principales métodos se encuentran:

- Método de Análisis de Arquitectura de Software
- Método de Revisión Intermedio de Diseño
- Método de Análisis de Acuerdos de Arquitectura de Software

3.4.1 Método de Análisis de Arquitectura de Software

El método de análisis de Arquitectura de Software (por sus siglas en inglés SAAM) fue el primero ampliamente documentado entre los métodos de evaluación. El mismo fue originalmente creado para el

análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida otros atributos de calidad, tales como portabilidad, escalabilidad e integralidad. Su utilización se basa en seis pasos básicos: se desarrollan los escenarios, se describe la arquitectura, se priorizan los escenarios y evalúan y finalmente se emite una evaluación global de los mismos.

La aplicación del método arroja resultados relacionados mayormente con la proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema. Además, evalúa el entendimiento de la funcionalidad del sistema y permite la comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

Este método si su objetivo es evaluar una sola arquitectura arroja los lugares donde la misma puede fallar y si se está en presencia de varias arquitecturas candidatas produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

3.4.2 Método de Revisión Intermedio de Diseño

El método de revisión intermedio de diseño (por sus siglas en inglés ARID) permite en las fases tempranas del diseño realizar una evaluación cualitativa basada en escenarios que permite analizar las debilidades en cuanto a riesgos de la arquitectura. Representa un híbrido que posee lo mejor del Análisis de Acuerdos de Arquitectura de Software (por sus siglas en inglés ATAM) combinado con el método Revisiones de Diseño Activa (por sus siglas en inglés ADR), éste último utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Su finalidad gira en torno a la calidad y la completitud y suficiencia de la documentación, el ajuste y la conveniencia de los servicios que provee el diseño propuesto. Es conveniente su utilización cuando se posee un diseño parcialmente completo que se desea evaluar

Este método arroja resultados importantes para evaluar la arquitectura dentro de los que se encuentran:

- La arquitectura en enfoques documentados.
- El conjunto de escenarios y sus prioridades.

- El árbol de utilidad.
- Los riesgos descubiertos.
- Los no riesgos documentados.
- Los puntos de sensibilidad y los acuerdos de puntos encontrados.

3.4.3 Método de Análisis de Acuerdos de Arquitectura de Software

El método de Análisis de Acuerdos de Arquitectura de Software (por sus siglas en inglés ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados y apoya a los involucrados con el proyecto a entender las consecuencias de las decisiones arquitectónicas tomadas con respecto a los atributos de calidad.

Este método puede aplicarse en fases tempranas del desarrollo del software. Los implicados para evaluar la arquitectura mediante este método son: un equipo de evaluación, otro equipo que posee el poder de decisiones sobre la arquitectura y por ultimo un equipo cuyos miembros registran el interés en la arquitectura, cada equipo juega un rol y realiza actividades con el fin evaluar los atributos de calidad. Su desarrollo se basa en una serie de fases fundamentales donde se desarrollan las actividades.

Los pasos de aplicación se muestran a continuación:

Fase 1: Presentación	
Presentación del ATAM.	1. El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.
Presentación de las metas del negocio.	2. Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.

Presentación de la arquitectura.	3. El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
Identificación de los enfoques arquitectónicos	4. Estos elementos son detectados, pero no analizados.
Generación del Utility Tree.	5. Se elicitan los atributos de calidad que engloban la “utilidad” del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
Análisis de los enfoques arquitectónicos	6. Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Fase 3: Reporte	
Lluvia de ideas y establecimiento de prioridad.	7. Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
Análisis de los enfoques arquitectónicos	8. Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso. 9. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Resultados	

Presentación de los resultados	10. Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Tabla 4. Pasos del método ATAM (8)

3.5 Evaluación de la Arquitectura Propuesta

Para evaluar la arquitectura de software que se propone, el método más conveniente a utilizar es el **ATAM**, este método es considerado el más completo porque revela la forma en que una arquitectura específica satisface los atributos de calidad seleccionados y provee una visión de cómo estos interactúan con otros. Este método presenta como principal desventaja que solo se utiliza después que la arquitectura ha sido diseñada, como es el caso. No sirve para comparar dos arquitecturas candidatas, pero dado a que no es el proceso que se quiere llevar a cabo; no constituye (desventaja) ningún inconveniente para iniciar la evaluación.

Además se propone utilizar la técnica de evaluación basada en **simulación**, su enfoque básico es la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse (9). Tiene como objetivo evaluar el comportamiento de la arquitectura bajo diversas circunstancias, una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad, pudiéndose mezclar con la técnica basada en **escenarios**, la cual es poco costosa para el equipo de desarrollo.

El ATAM, como se explicó anteriormente está compuesto por una serie de actividades estructuradas en fases, después del diseño y la descripción realizada en el **Capítulo 2**; quedó claro todo el proceso del negocio y el objetivo que se persigue con la arquitectura, por lo que se pasa a la fase 2 (ver **Tabla 5**), donde se especificarán los atributos con sus escenarios correspondientes.

Atributos de Calidad	Perfil	Escenarios
Funcionalidad	Operaciones sobre el plug-in.	▪ Operaciones Desinstalar, Habilitar y Deshabilitar Plug-in. ▪ Operación Importar Plug-in.
Portabilidad	Portabilidad	▪ Existe la necesidad de cambiar la plataforma sobre la cual corre el sistema.
Reusabilidad	Reusabilidad	▪ Los plug-in deben ser reutilizables en futuras aplicaciones.
Integrabilidad	Integrabilidad	▪ Se desea que los plug-in importados funcionen correctamente.
Escalabilidad	Ampliación	▪ Se desea que el diseño arquitectónico pueda ser ampliado.

Tabla 5. Árbol de utilidad

Para hacer factible el entendimiento de la técnica de evaluación escogida, es necesario concebir una interfaz que dé una idea de lo que se quiere lograr con la arquitectura diseñada anteriormente, constituyendo uno de los pasos a seguir para una fructífera evaluación mediante la técnica seleccionada, ver **Figura 13**.

Para la realización de la simulación se hizo necesario extender el diagrama de clases con el objetivo de adaptar la arquitectura al framework Qt, **Ver anexo 5**. La misma representa la implementación de las funcionalidades más significativas donde se muestra la operación que se puede realizar sobre un plug-in cualquiera, quedando demostrada la genericidad de la Arquitectura propuesta.

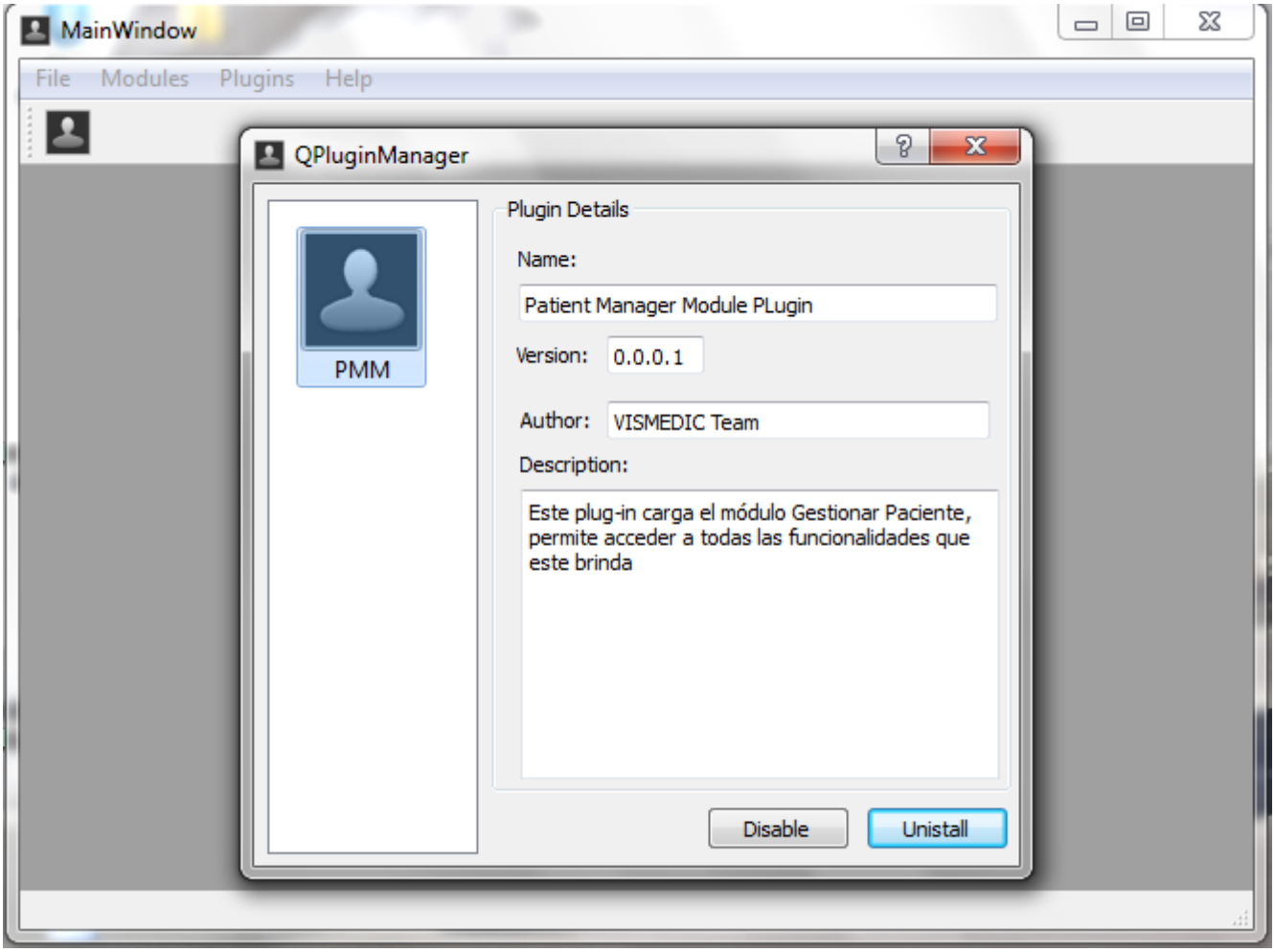

Figura 13. Prototipo funcional de la simulación de la arquitectura

Este prototipo funcional provee al usuario interactuar con una interfaz intuitiva y fácil de manejar, puesto que las funcionalidades que pueden ser ejecutadas son de fácil acceso como se muestra en la figura anterior.

A continuación se pasa a la evaluación de cada uno de los atributos definiendo primeramente los escenarios, a partir del comportamiento de cada uno de ellos en la simulación, escogidos para demostrar que la arquitectura cumple con las expectativas del equipo de desarrollo.

Definición de los principales escenarios:

Los principales escenarios serán aquellos que tengan según el arquitecto una prioridad alta y además poseen una alta dificultad de implementación.

Fuente del estímulo	Se trata de alguna entidad, ser humano, sistema u otra entidad que interactué con el sistema.
Estímulo	Condición que necesita ser considerada cuando arriba al sistema.
Entorno	El estímulo se produce cuando el sistema se encuentra bajo ciertas condiciones que se toman en cuenta para valorar el atributo de calidad evaluado.
Componentes o Artefactos	Son aquellos componentes que son afectados bajo el escenario representado
Respuesta	Actividad realizada después de que el sistema es estimulado.
Medida de la Respuesta	Cuando el sistema emita alguna respuesta ésta debe ser medida de alguna manera para que el requerimiento pueda ser aprobado.

Tabla 6. Definición general de un escenario (10)

- **Escenario #1: Operaciones (Desinstalar, Habilitar y Deshabilitar) sobre el plug-in**

Fuente del estímulo	El usuario desea operar sobre el plug-in Desinstalar, Habilitar y Deshabilitar.
Estímulo	Evento clic del mouse sobre cualquiera de las operaciones disponibles.
Entorno	La funcionalidad se ejecuta normal bajo las siguientes condiciones: tarjeta gráfica de 512 MB con soporte para textura 3D, un procesador Pentium 4 a 3.0 GHz o superior, una memoria RAM de 1GB de capacidad y espacio en disco duro

	de 512 MB.
Componentes o Artefactos	Todo el sistema
Respuesta	El sistema ejecuta cualquiera de las funcionalidades antes mencionadas.
Medida de la Respuesta	El sistema queda de la siguiente forma: ▪ Se elimina un plug-in del sistema. ▪ Se habilita un plug-in en el sistema anteriormente deshabilitado. ▪ Se desactivan las funcionalidades de un plug-in.

Tabla 7. Escenario Desinstalar, Habilitar y Deshabilitar de un plug-in

- **Escenario #2: Importar plug-in**

Fuente del estímulo	El usuario desea cargar un plug-in.
Estímulo	Evento clic del mouse sobre la opción Importar.
Entorno	La funcionalidad se ejecuta normal bajo las siguientes condiciones: tarjeta gráfica de 512 MB con soporte para textura 3D, un procesador Pentium 4 a 3.0 GHz o superior, una memoria RAM de 1GB de capacidad y espacio en disco duro de 512 MB.
Componentes o Artefactos	Todo el sistema
Respuesta	El sistema importa el plug-in, si el plug-in importado depende de otro no se puede hacer usos de sus funcionalidades.

Medida de la Respuesta	Se ha cargado un nuevo plug-in al sistema.

Tabla 8. Escenario Importar Plug-in

- **Escenario #3: Portabilidad**

Fuente del estímulo	Se desarrolla el núcleo de la aplicación en un sistema operativo diferente de Windows dígase Linux.
Estímulo	Se desea utilizar las facilidades de los plug-in en el sistema.
Entorno	Ejecución normal en el sistema operativo Linux
Componentes o Artefactos	Todo el sistema.
Respuesta	El sistema funciona correctamente.
Medida de la Respuesta	En el sistema operativo Linux se ejecutan correctamente todas las funcionalidades del sistema.

Tabla 9. Escenario Portabilidad

- **Escenario #4: Reusabilidad**

Fuente del estímulo	Otro sistema requiere del diseño arquitectónico propuesto para su desarrollo.
Estímulo	Se desea utilizar diseño arquitectónico propuesto.
Entorno	Ejecución normal de acuerdo al sistema que quiera hacer uso del diseño arquitectónico propuesto.
Componentes o	El diseño arquitectónico de plug-in.

Artefactos	
Respuesta	Otro sistema puede reutilizar el diseño arquitectónico propuesto.
Medida de la Respuesta	El uso del diseño arquitectónico propuesto hará que el sistema que lo utilice funcione correctamente, siempre y cuando no exista dependencia entre los plug-in que se deseen cargar.

Tabla 10. Escenario Reusabilidad

- **Escenario # 5: Integrabilidad**

Fuente del estímulo	Se desea que los plug-in importados funcionen correctamente.
Estímulo	Se carga un plug-in a la aplicación.
Entorno	Ejecución normal
Componentes o Artefactos	Todo el sistema
Respuesta	El plug-in importado funciona correctamente.
Medida de la Respuesta	El sistema ejecuta cada una de sus funcionalidades del plug-in correctamente, siempre y cuando no exista dependencia entre los plug-in que se deseen cargar.

Tabla 11. Escenario Integrabilidad.

- **Escenario #6: Ampliación**

Fuente del estímulo	Se quiere adaptar el diseño arquitectónico a un IDE específico.
Estímulo	Se desea ampliar el diseño arquitectónico.

Entorno	Ejecución normal
Componentes o Artefactos	Todo el sistema
Respuesta	El diseño arquitectónico brinda al sistema las mismas facilidades; la ampliación realizada debe seguir la estructura definida.
Medida de la Respuesta	El sistema sigue ejecutándose correctamente.

Tabla 12. Escenario Ampliación

A partir de la especificación de cada uno de los atributos de calidad definidos en la propuesta en la propuesta de solución se puede concluir:

Riesgos: A pesar de que la aplicación en cuestión no requiere de módulos que dependan de otro, constituye un riesgo y una desventaja el hecho de que no se pueda ejecutar un plug-in que tenga dependencia con otro, aspecto identificado en los escenarios: Integrabilidad y Reusabilidad.

Táctica y recomendación: Investigación sobre cómo llevar a cabo la implementación y la gestión de la dependencia entre plug-in.

No riesgos: En cuanto a las funcionalidades previstas en el diseño de la arquitectura propuesta la simulación realizada no presenta ningún inconveniente permitiendo **Importar, Desinstalar, Deshabilitar y Habilitar** un plug-in, mientras se cumpla con las restricciones arquitectónicas descritas en el capítulo anterior dando respuesta rápida a cada uno de estos eventos, demorando un tiempo aproximado de 2 segundos, obteniendo de esta forma un mayor rendimiento de la aplicación.

El diseño arquitectónico está previsto para que pueda ser ejecutado tanto en Linux como en Windows sólo sería necesario reutilizar el código de la clase DinamycPluginHandle implementada para Windows y cambiarle la extensión (*.so para Linux y *.dll para Windows) del archivo que se desea cargar.

La extensibilidad del sistema demuestra que la necesidad de agregar funcionalidades en tiempo de ejecución al sistema, queda solventada. Se brinda la posibilidad de cargar los módulo que se desean utilizar e incluso tenerlos activos o no, según las necesidades del usuario.

Cada uno de los plug-in que utiliza la aplicación puede ser reutilizado por otros sistemas con fines similares a Vismedic.

3.6 Consideraciones parciales

En este capítulo a partir de la valoración de los distintos atributos de calidad que respondían al desarrollo de la arquitectura propuesta, así como de los métodos de evaluación se llegaron a los siguientes resultados:

- Se seleccionó como método de evaluación: Método de Análisis de Acuerdos de Arquitectura de Software, con el uso de los siguientes atributos de calidad: funcionalidad, portabilidad, reusabilidad, integrabilidad y escalabilidad.
- La técnica de evaluación escogida fue la simulación.
- Se identificó un riesgo en la aplicación de estos métodos él cual no representa una amenaza significativa para el fin de la arquitectura propuesta.

CONCLUSIONES

Al finalizar la investigación y después de haber valorado cada uno de los resultados obtenidos se pudo arribar a las siguientes conclusiones:

- A partir del estudio realizado se logró definir la arquitectura permitiendo obtener un sistema extensible y configurable, mediante la carga en tiempo de ejecución de las funcionalidades que el usuario, que interactúa con la aplicación, necesita en un momento determinado.

- La validación de la arquitectura permitió conocer cómo se comportará el sistema ante las acciones del usuario, permitiendo afirmar que la arquitectura propuesta cumple con los requerimientos funcionales y no funcionales que debe poseer el Sistema de Visualización Médica, Vismedic.

- La idea a defender planteada como propuesta de solución inmediata al problema científico de la investigación, ha quedado validada de manera satisfactoria a través de los resultados de la descripción de la arquitectura y el comportamiento de los atributos de calidad de la solución propuesta, dándole cumplimiento de esta manera al objetivo general de la investigación.

RECOMENDACIONES

Se recomienda:

- Investigar profundamente el tema de la dependencia entre plug-in con el objetivo de satisfacer la reutilización de la Arquitectura propuesta en caso de que otro sistema lo requiera.
- Aplicar otros métodos y técnicas de evaluación de la arquitectura para corroborar y ampliar los resultados obtenidos.

BIBLIOGRAFÍA REFERENCIADA

1. Reynoso, Carlos Billy.*Introducción a la Arquitectura de Software.* 2005.

2. UCI, Departamento Ingenieria. EVA. [En línea] 2010. [Citado el: 10 de 10 de 2010.] http://eva.uci.cu/mod/resource/view.php?id=14075.

3. Buschmann. *PATTERN-ORIENTED SOFTWARE ARCHITECTURE, A system of Patterns.* 2000.

4. Reynoso.*Introducción a la Arquitectura de Software. Volume, 1-10.* 2004.

5. Bosch.*Design Patterns as Language Constructs: Journal of Object Oriented Programming (JOOP).* 1998.

6. Johannes Mayer, Ingo Melzer, Franz Schweiggert.*Ligthweight Plug-in Based Aplication Development.*

7. Azad, Bolour. Eclipse. [En línea] Informática Bolour, 3 de 7 de 2003. [Citado el: 21 de 3 de 2011.] http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html.

8. Ayala, Dayami Chávez. *Arquitectura de la Plataforma de Transmisión Abierta para Radio y Televisión.* Ciudad de La Habana : s.n., 2010.

9. ERIKA CAMACHO, FABIO CARDESO,GABRIEL NUÑEZ. *ARQUITECTURAS DE SOFTWARE.* 2004.

10. Cid., Reynier Arias. *Propuesta de arquitectura de un subsistema de modelado de Diagramas de Flujo de Información.* 2009.

BIBLIOGRAFÍA CONSULTADA

1. Adrián Lasso, MVP. Scribd. [En línea] 2010. [Citado el: 2010 de 12 de 12.] http://www.scribd.com/doc/210452/Arquitectura-de-Software-Adrian-Lasso.

2. Kruchten P.*Architectural Blueprints—The "4+1" View Model of Software Architecture. IEEE Software, pp.42-50.* 1995. en línea http://www.computer.org/portal/web/csdl/doi/10.1109/52.469759 citado el 17/02/2011.

3. Qt. [En línea] Nokia Corporation , 2010. [Citado el: 21 de 03 de 2011.] http://doc.qt.nokia.com/4.6/plugins-howto.html.

4. Brooks, Jr Frederick.*The mythical man-month. s.l.* s.l. : Addison-Wesley, 1975.

5. Perry, Dewayne y Wolf, Alexander.*Foundations for the study of software architecture. s.l.* s.l. : ACM SIGSOFT Software Engineering, 1992.

6. Reynoso, Carlos Billy. Introducción a la Arquitectura de Software,Versión 1.0. [En línea] Marzo de 2004. [Citado el: 2 de 12 de 2010.] http://eva.uci.cu/mod/resource/view.php?id=14075.

7. Gutiérrez, Javier J. Lenguajes y Sistemas Informáticos. [En línea] 2006. [Citado el: 11 de 12 de 2010.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

8. [En línea] irección Nacional de Servicios Académicos Virtuales 2011, 2011. [Citado el: 2011 de 1 de 7.] http://www.virtual.unal.edu.co/cursos/sedes/manizales/4060010/lecciones/Capitulo1/modelo.htm.

9. Reynoso, Carlos y Kicillof, Nicolás.*Lenguajes de Descripción de Software.* 2004.

10. Dávila, Jose A Vela. Ponencia en Seminario. [En línea] 2008. [Citado el: 10 de 02 de 2011.] http://www.cimat.mx/Eventos/seminariotecnologias08/javd.pdf-.

11. Kiccillof, Carlos Reynoso – Nicolás. WiiliDev. [En línea] Marzo de 2004. [Citado el: 2010 de 11 de 11.] Estilos y Patrones en la Estrategia de Arquitectura de Microsoft en línea http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

12. —. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* s.l. : en linea http://www.willydev.net/descargas/prev/Estiloypatron.pdf, Versión 1.0, 2004.

13. Salanitri, Sergio. Arquitectura de Software, programación , tendencias y reflexiones personales. [En línea] 21 de 07 de 2009. [Citado el: 13 de 12 de 2010.] http://ssalanitri.blogspot.com/2009/07/plugin-pattern.html.

14. johannes MAyer, Ingo Melzer, Franz.*Lightweigh Plug-in-Based Aplication Depelopment.* Alemania : Department of Aplied Information Proceessing, University of Ulm.

15. Woodwing. [Online] 2011. [Cited: 4 de 4 de 2011.]

http://www.woodwing.nl/es/enterprise-publishing-system/extensibility.

16. Yoan Arlet Carrascoso Puebla, Enrique Chaviano Gómez y Anisleydi Céspedes Vega. GestioPolis. [Online] 5 7, 2009. [Cited: 4 5, 2011.] http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.

17. Cid., Reynier Arias. *Propuesta de arquitectura de un subsistema de modelado de Diagramas de Flujo de Información.* 2009.

ANEXOS

Anexo 1: Caso de uso Importar plug-in

Caso de Uso:	Importar plug-in	
Actores:	Médico	
Resumen:	El caso de uso inicia cuando el médico selecciona la opción importar, el sistema le muestra una ventana de búsqueda y el médico selecciona un plug-in y el sistema lo coloca en la lista de plug-in importados.	
Precondiciones:	Debe existir al menos un plug-in en el directorio.	
Referencias:	RF1	
Prioridad:	Crítico	
Flujo Normal de Eventos		
Sección "Principal"		
Acción del Actor		**Respuesta del Sistema**
1. El médico accede al menú Plugins y selecciona la opción Importar plug-in.		1.1 El sistema muestra una ventana de búsqueda.
2. El médico selecciona el plug-in a importar.		2.1 El sistema muestra el plug-in importado con todos sus datos (nombre, versión, autor, descripción).
Poscondiciones:	El sistema muestra el plug-in importado.	

Anexo 2: Caso de Uso Desinstalar plug-in

Caso de Uso:	Desinstalar plug-in	
Actores:	Médico	
Resumen:	El caso de uso inicia cuando el médico desea desinstalar un plug-in, el sistema muestra una ventana, el médico selecciona el plug-in y presiona la opción desinstalar y el sistema elimina el plug-in de la lista de plug-in activos.	
Precondiciones:	El plug-in debe estar Importado.	
Referencias:	RF2	
Prioridad:	Crítica	
Flujo Normal de Eventos		
Sección "Principal"		
Acción del Actor		**Respuesta del Sistema**
1.El médico accede a la ventana Administrar plug-in.		1.1 El sistema muestra el listado de plug-in importados.
2. El médico selecciona el plug-in que desea desinstalar y presiona el botón "delete".		2.1 El sistema elimina de la lista de plug-in activos el componente seleccionado.
Poscondiciones:	Queda eliminado el plug-in seleccionado del sistema.	

Anexo 3: Caso de Uso Habilitar plug-in

Caso de Uso:	Habilitar plug-in	
Actores:	Médico	
Resumen:	El caso de uso inicia cuando el médico desea habilitar un plug-in, el sistema muestra una ventana con los plug-in importados, el médico selecciona el plug-in y presiona la opción habilitar y el sistema habilita la funcionalidad que brinda el plug-in.	
Precondiciones:	El plug-in debe estar deshabilitado en el sistema.	
Referencias:	RF4	
Prioridad:	Crítico	
Flujo Normal de Eventos		
Sección "Principal"		
Acción del Actor		**Respuesta del Sistema**
1. El médico accede a la ventana Administrar plug-in.		1.1 El sistema muestra el listado de plug-in importados.
2. El médico selecciona el plug-in que desea habilitar y presiona el botón "enable".		2.1 El sistema habilita las funcionalidades del plug-in seleccionado.
Poscondiciones:	Se activan en el sistema las funcionalidades que brinda el plug-in.	

Anexo 4: Caso de Uso Deshabilitar plug-in

Caso de Uso:	Deshabilitar plug-in.
Actores:	Médico
Resumen:	El caso de uso inicia cuando el médico desea deshabilitar un plug-in, el sistema muestra una ventana con los plug-in importados, el médico selecciona el plug-in que desea deshabilitar y presiona la opción deshabilitar, el sistema deshabilita las funcionalidades del plug-in.
Precondiciones:	El plug-in debe estar importado.
Referencias:	RF5
Prioridad:	Alta

Flujo Normal de Eventos	
Sección "Principal"	
Acción del Actor	**Respuesta del Sistema**
1. El médico accede a la ventana Administrar plug-in.	1.1 El sistema muestra el listado de plug-in importados.
2. El médico selecciona el plug-in que desea deshabilitar y presiona el botón "desable".	2.1 El sistema deshabilita las funcionalidades del plug-in seleccionado.
Poscondiciones:	Se deshabilitan todas la funcionalidades que brinda el plug-in.

Anexo 5: Diagrama extendido

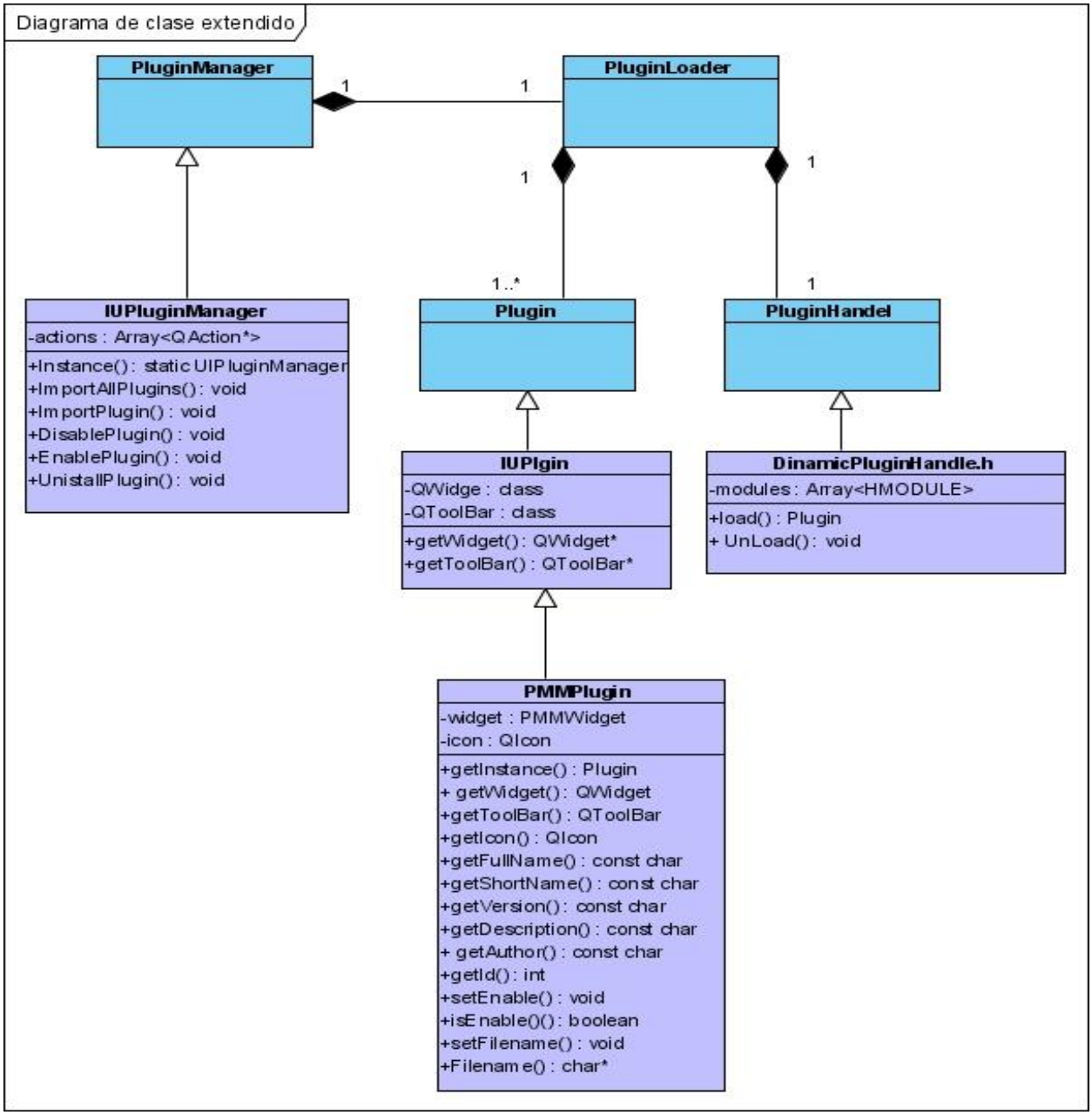

Figura 13 Diagrama de clase extendido

GLOSARIO DE TÉRMINOS

Componente: Recursos desarrollados para un fin concreto y que puede formar solo o junto con otros, un entorno funcional requerido por cualquier proceso predefinido.

Evaluar: Determinar, estimar el valor, el precio o la importancia de algo.

Framework: Es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

IDE: Entorno de desarrollo integrado. Conjunto de programas que se ejecuta a partir de una única interfaz de usuario. Por ejemplo, los lenguajes de programación incluyen a menudo un editor de texto, un compilador y un depurador, los cuales pueden activarse y funcionar a partir de un menú común.

Línea base: Es el cimiento a partir del cual se pueden establecer las pautas y pasos a seguir para implantar una Arquitectura de Software.

Sistema: Consiste en programas informáticos que sirven para controlar e interactuar con el sistema operativo, proporcionando control sobre el hardware y dando soporte a otros programas.

Técnica: Es un procedimiento o conjunto de reglas, normas o protocolos, que tienen como objetivo obtener un resultado determinado, ya sea en el campo de la ciencia, de la tecnología, del arte, del deporte, de la educación o en cualquier otra actividad.

Vista: Presentación de un modelo, la cual es una descripción completa de un sistema desde una perspectiva particular.

Visualización: Consiste en programas informáticos que sirven para controlar e interactuar con el sistema operativo, proporcionando control sobre el hardware y dando soporte a otros programas.