Universidad de las Ciencias Informáticas

Facultad 7

Aplicación para la Integración y Comunicación de módulos del alas PACS en una Estación de Visualización: alas PACSManager

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

Autores: Adrian Font Hernández

Ariel Martínez Roque

Tutores: Yoel Rivera Suarez

Luis Javier Vallejo Mursulí

La Habana, junio de 2011
"Año 53 de la Revolución"

DATOS DE CONTACTO

Tutores:

Ing. Yoel Rivera Suarez
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email:yrsuarez@uci.cu

Ing. Luis Javier Vallejo Mursulí
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: ljvallejo@uci.cu

RESUMEN

Con el presente trabajo se desarrolla una solución de software que garantiza la gestión de componentes en una **estación de visualización** del sistema **alas PACS**, sirviendo como principal orquestador entre los mismos componentes. La solución adopta el nombre de desarrollo: **alas PACSManager** y es desarrollada por el CESIM en la Universidad de las Ciencias Informáticas (UCI).

Para el desarrollo de este software se utilizó como Entorno Integrado de Desarrollo (IDE), Visual Studio 2010; como marco de trabajo **.NET Framework 4** y como lenguaje de programación C-Sharp. Se realizó un estudio sobre las principales librerías existentes para la comunicación entre sistemas dentro de .NET Framework 4.

El alas PACSManager garantiza la comunicación entre los componentes del alas PACS a través de mensajes, dando respuestas a las peticiones hechas desde cualquier componente de una estación de visualización del alas PACS. Además permite la comercialización de los módulos de la solución alas PACS por separado.

Palabras claves:

Estación de visualización, alas PACS, alas PACSManager, .NET Framework 4.

INDICE

Introducción:

Durante los primeros años de la era de las computadoras estas eran en sí sólo un conglomerado de componentes electrónicos, conocidos hoy como hardware. Sin embargo, para que estos componentes funcionaran de forma ordenada y a la vez poder aprovechar todos los recursos que esta brindaba de manera que se pudiera explotar al máximo su capacidad, se creó el software. El cual, está formado por una serie de instrucciones lógicas y datos, que permiten la interacción usuario y máquina.

En sus inicios, los software eran pequeños y satisfacían solo objetivos específicos; la mayoría se desarrollaban y eran utilizados por la misma persona u organización. La misma persona lo escribía, lo ejecutaba y, si fallaba, lo depuraba. El diseño era un proceso implícito, realizado en la mente de alguien y, la documentación normalmente no existía. El desarrollo del software se realizaba virtualmente sin ninguna planificación, hasta que los planes comenzaron a romperse y los costes a correr. Los programadores trataban de hacer las cosas bien, y con un esfuerzo heroico, a menudo salían con éxito. El software se diseñaba a medida para cada aplicación y tenía una distribución relativamente pequeña, aun así ganaban en tiempo y recursos y empezaron a adentrarse en la mayoría de negocios y procesos comerciales de la vida social del hombre. [1]

Esta crecida propició un auge en la variedad de los software, los mismos eran cada vez más complejos ya que los flujos de trabajos eran muy variables en la mayoría de los centros laborables y estos querían informatizar más procesos de su gama de servicios. Para dar soporte a estos software e incrementar las nuevas prestaciones surgieron nuevas tecnologías como fueron los lenguajes de programación de alto nivel cada vez con más rutinas y librerías implementadas, cada uno con propósitos y formas distintas. No obstante, todos estos lenguajes pueden ser clasificados en una jerarquía de familias a partir del modelo que siguen para definir y operar información, es decir, que pueden ser jerarquizados según el paradigma (del latino paradigma, del griego paradigma, significado Modelo u /o ejemplo) [2] que siguen.

Según Robert Floyd los paradigmas de programación son un proceso de diseño que va más allá de una gramática, reglas semánticas y algoritmos, en otras palabras es un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas. Estos paradigmas brindan numerosas ventajas como: la posibilidad de subdividir una aplicación en partes más pequeñas (módulos), aumentar la cohesión de los módulos de un sistema cada una de las cuales debe ser tan independiente como sea posible de la

aplicación en sí y de las restantes partes (estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos), entre otras.

Era importante que los programas fueran cada vez más flexible ya que aparecían nuevas plataformas sobre las cuales serían ejecutados además de nuevas versiones y también que fueran más extensibles ya que cada vez era más necesaria la incorporación de nuevas funcionalidades y servicios. Esto trajo como consecuencia la aparición de problemas recurrentes sin importar el ámbito para el cual había sido creado el programa. Para dar solución a este tipo de problemas surgen los patrones de software.

Los patrones de software describen un problema que ocurre una y otra vez en un entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo dos veces de la misma forma.

Son un esquema de solución que se aplica a un tipo de problema, la cual no es mecánica, sino que requiere de adaptación y matices. Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares, además facilitan la reutilización del diseño y de la arquitectura, capturando las estructuras estáticas y dinámicas de colaboración de soluciones exitosas a problemas que surgen al construir aplicaciones. Los patrones le ayudan a construir sobre la experiencia colectiva de ingenieros de software experimentados. Estos capturan la experiencia existente y que ha demostrado ser exitosa en el desarrollo de software, y ayudan a promover las buenas prácticas de diseño. Cada patrón aborda un problema específico y recurrente en el diseño o implementación de un sistema de software. [3]

Existen muchos patrones pero se deben tener presente siempre los siguientes elementos de un patrón, denominado reglas de tres partes:

- ✓ Su nombre.
- ✓ El problema (cuando aplicar un patrón).
- ✓ La solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

Con estas nuevas técnicas, a la hora de crear un programa, los desarrolladores podían continuar con la labor que otros habían comenzado antes; así surgieron los equipos de desarrollo, las empresas de desarrollo de software, entre otros, los cuales tenían como fin informatizar, actualizar, dar soporte y consultoría a empresas clientes.

El software comenzaría a utilizarse en varios procesos de negocios e industriales y a inmiscuirse en la mayoría de las ramas y sectores de la sociedad hasta el punto de convertirse en una ciencia y crear nuevas formas de ciencia como es el caso de la informática Médica que es la aplicación de informática en la salud mediante el uso de software. Su principal objetivo es prestar servicio a los profesionales de la salud para mejorar la calidad y la eficacia de la atención médica de la sociedad.

Para la salud se han creado varios software que satisfacen las necesidades y servicios de la misma, tal es el caso de los sistemas de registros médicos o historias clínicas, gestión de documentación, administración de datos, gestión de sistemas hospitalarios, software de asistencia médica para el diagnóstico de pacientes, análisis de datos e imágenes médicas, etc. Uno de los objetivos del perfeccionamiento de los sistemas de salud actualmente es lograr un mejor aprovechamiento de los recursos humanos, el conocimiento, las técnicas y equipos que se disponen para ofrecer una atención cada vez más rápida y especializada a los pacientes, sin que la distancia constituya una limitación para lograrlo. Las industrias desarrolladoras de software han jugado un papel importante para la salud, con el fin de cubrir estas necesidades con un alto nivel, una calidad de excelencia y una seriedad absoluta, ya que en un simple fallo en cualquier entorno informático podría costar la vida a un ser humano.

La explotación eficiente de los equipos de imágenes médicas (tomografía axial computarizada, resonadores magnéticos, eco-cardiógrafos, angiógrafos, video-endoscopios, entre otros) requiere disponer de un conjunto de equipos informáticos dedicados a la adquisición, almacenamiento, procesamiento y comunicación de imágenes radiológicas digitales e información asociada conocido mundialmente como PACS (Picture Archiving and ComunicationSystem).

Los PACS constituyen el principal avance en la gestión automatizada de la información en los departamentos de diagnóstico por imágenes. Su principal objetivo es permitir el funcionamiento del servicio de imágenes sin la necesidad de la impresión de placas radiológicas ni de papeles para la información clínica asociada a las imágenes, supliendo los altos costos en tiempo y dinero que esto significa. Tienen además la capacidad de interactuar con otras aplicaciones clínicas como el sistema de Información Hospitalaria (HIS) o el Sistema de Información Radiológica (RIS). [4]

Por otro lado Cuba no ha estado al margen de este interés mundial y ha volcado gran parte de su desarrollo y recursos a la informatización de algunos de sus renglones económicos. La Universidad de las Ciencias

Informática (UCI) como pilar fundamental de este proceso ha concentrado su fuerza de trabajo en la informatización primeramente de los sectores principales de la economía de Cuba, como es la salud, para la cual se desarrolló un PACS, con el nombre alas PACS, por el Centro de Informática Médica: (CESIM).

Este software médico ha ido evolucionando en sus distintas fases de pruebas ya que en las clínicas y hospitales en que se ha desplegado, los flujos de trabajos son muy variables y cada vez estas instituciones tratan de incorporar nuevos servicios. El equipo de desarrollo ha presentado problemas a la hora de implementar nuevas versiones o funcionalidades por la arquitectura poco flexible que presenta dicha solución, además de presentar problemas de bajo acoplamiento. Estos problemas han traído consigo la no comercialización de los módulos del alas PACS por separado, teniendo que comprar los clientes la solución íntegra.

En la actualidad existen nuevas tecnologías que propician la extensibilidad de soluciones integrales, así como la eliminación de los problemas de acoplamiento y cohesión, permitiendo la incorporación de componentes dinámicamente y la integración con otras aplicaciones, estas tecnologías brindan soluciones a los problemas por los cuales ha transcurrido el equipo de desarrollo del CESIM. Algunas de ellas son: Managed Extensibility Framwork (MEF), Remoting/IPC, Windows Communication Fundation(WCF) y algunos servicios como son Microsoft Biztalk Server.

Por todo lo anterior, se plantea como **situación problémica:** el bajo acoplamiento existente entre los módulos del alas PACS, así como una arquitectura poco flexible y una baja reutilización de componentes, los cuales propician un alto costo en tiempo y esfuerzo a la hora de realizar una modificación o la inclusión de una nueva funcionalidad al sistema alas PACS. Por ello se define el siguiente **problema a resolver:** ¿Cómo administrar la interacción entre los componentes que integran la solución alas PACS en una estación de visualización?

El **objeto de estudio:** Tecnologías y técnicas .NET para la interacción entre componentes y el **campo de acción:** tecnologías y técnicas .NET para la interacción entre componentes que integran una estación de visualización del alas PACS. Para darle solución al problema planteado se define como **objetivo general:** implementar un componente de software que logre la integración, configuración y manejos de los componentes del alas PACS.

Para lograr el correcto cumplimiento del objetivo se proponen las siguientes **tareas investigativas:**

- ✓ Valorar las tecnologías y mecanismos existentes para el desarrollo de componentes para la gestión de la comunicación de módulos de un sistema informático.
- ✓ Realizar el modelado del dominio del alas PACSManager.
- ✓ Identificar los requisitos de software para la implementación del alas PACSManager.
- ✓ Realizar el modelado y descripción de los casos de uso del sistema.
- ✓ Definir la arquitectura para el alas PACSManager.
- ✓ Realizar análisis y diseño del alas PACSManager.
- ✓ Realizar la implementación y las pruebas del alas PACSManager.

Capítulo 1: Fundamentación Teórica: Contiene un estudio del estado del arte de las principales tecnologías que existen en el mundo para la comunicación entre sistemas. Análisis de las diferentes tecnologías que se pueden emplear en la solución, con vistas a seleccionar las más apropiadas para llevar a cabo dicho proceso con éxito y calidad en el desarrollo de la investigación.

Capítulo 2: Características del Sistema: En el capítulo, como parte de la propuesta de solución, se describen las características del sistema, la especificación de los requisitos funcionales y no funcionales del componente alas PACSManager, determinándose a su vez los casos de uso y los actores. Para cada uno de los casos de uso contiene una descripción del mismo.

Capítulo 3: Diseño del Componente: Contiene lo referente al análisis y diseño del componente. Como parte de la solución se modelan los diagramas de clases del análisis, los diagramas de clases del diseño, así como los diagramas de interacción correspondientes.

Capítulo 4: Implementación del Componente: Aborda todo lo relacionado con el flujo de trabajo de implementación. En el mismo se especifican los componentes ejecutables del componente alas PACSManager, así como su interacción. Se exponen además los diagramas de implementación

CAPÍTULO 1. Fundamentación Teórica

En este capítulo se abordará todo lo referente al estudio del arte de los servicios que proveen comunicación a los sistemas así como de las tendencias, técnicas y tecnologías usadas en la actualidad y que servirán de apoyo para la solución del problema al que se enfrenta. Todo con el objetivo fundamental de brindar información complementaria para propiciar un mejor entendimiento de los capítulos venideros.

Marco Teórico Conceptual

1.1 Estudio del Arte

El software complejo con un negocio muy amplio, se apoya en varios módulos para la distribución y desarrollo de las tareas y funciones, además de ser más factibles y comerciables. Una solución integrada por varios módulos proporciona flexibilidad para las soluciones a la hora de incluir nuevos servicios, un PACS, como solución de este tipo, permite la venta a varias clínicas y hospitales no importa que tan variables sean sus flujos de trabajo ya que el mismo está compuesto por varios módulos que se encargan a la vez de diversos componentes. Estos componentes pueden estar en distintas partes físicas incluso pueden haber varios en una misma estación de trabajo, los mismos necesitan interactuar y relacionarse en cualquier momento.

Para dar solución a este inquietante obstáculo surgen los orquestadores como es el caso de Microsoft BizTalk Server. Estos son los encargados de manejar toda la información que se tramita en las diversas unidades de un software; los mismos se basan particularmente en arquitectura, la cual puede estar orientada a servicios o extensibilidad.

Microsoft BizTalk Server

Microsoft BizTalk Server tiene características que simplifican la administración e implementación de las soluciones, este incluye un contenedor de aplicaciones de BizTalk para los elementos que componen una solución empresarial, como orquestaciones, esquemas, asignaciones, canalizaciones y ensamblados .NET. Se puede administrar, modificar, implementar e instalar todos los elementos de una aplicación como una sola unidad, además, posee asistentes que ayudan a automatizar las tareas de implementación de la aplicación. El motor de BizTalk Server es el componente principal del producto. Este cuenta con dos piezas principales: un componente de mensajería que proporciona la capacidad de establecer una comunicación

con otros tipos de software. Dado que está basado en adaptadores para distintos tipos de comunicación, el motor admite una amplia gama de protocolos y formatos de datos, como, por ejemplo, los servicios Web, entre otros y un soporte para la creación y ejecución de procesos definidos gráficamente, las orquestaciones. Las orquestaciones, integradas sobre los componentes de mensajería del motor, implementan la lógica que dirige un proceso empresarial completo o parte de éste. [5]

Algunos de los servicios de Biz Talk son: [5]

Asistente para importación. El Asistente para importación, disponible desde la consola de administración de BizTalk Server, se usa para importar artefactos desde un archivo .msi a una aplicación de BizTalk. Si la aplicación especificada no existe, el Asistente para importación la crea. El asistente también registra y almacena los artefactos en el archivo .msi en la base de datos de administración de BizTalk.

Asistente para instalación. Puede iniciar el Asistente para instalación como paso final del Asistente para importación. El asistente instala la aplicación en el equipo local para que pueda ejecutar la aplicación en él.

Asistente para exportación. El Asistente para exportación, también disponible desde la consola de administración de BizTalk Server, se usa para generar un archivo .msi desde una aplicación de BizTalk. Después, puede usar el Asistente para importación para importar el archivo .msi a un grupo de BizTalk diferente. Esto facilita la tarea de agregar recursos a una aplicación o bien crea de forma automática una aplicación en el nuevo grupo de BizTalk.

Herramienta de la línea de comandos BTSTask. BTSTask es compatible con las características de implementación de aplicaciones de BizTalk Server, por lo que se pueden realizar muchas operaciones desde la línea de comandos.

1.1.2 Ámbito Nacional

El alas PACS no cuenta actualmente con un componente que permita la integración de sus módulos. El Departamento de Software Médico Imagenológico (SWMI) de la Universidad de las Ciencias Informáticas ha asumido la tarea de desarrollar un servicio informático para dar respuesta a la necesidad de la gestión de las acciones entre los módulos del alas PACS en una estación de visualización ya que las versiones anteriores presentan una arquitectura poco flexible que imposibilita que la solución alas PACS se pueda

adecuar fácilmente a cambios o inclusiones de nuevos servicios, además de su no comercialización por separado.

1.1.3 Análisis crítico de las soluciones existentes:

A pesar de las ventajas de BizTalk Server, este es un servicio usado solamente en ambientes empresariales utilizado para soluciones muy amplias que utilizan una inmensa cantidad de módulos, lo que trae consigo un consumo de recursos innecesario, además es un software privativo.

Para el desarrollo del alas PACSManager se tendrá en cuenta la arquitectura y las facilidades de BizTalk Server adaptándolas y enfocándolas a las necesidades actuales del sistema alas PACS, entre las cuales se pueden mencionar la mayor flexibilidad ante cambios y la eliminación del alto acoplamiento entre sus módulos. Una muestra de esto en BizTalk Server es su capacidad de administración, modificación, implementación e instalación de todos los elementos de una aplicación como una sola unidad. Además de su asistente para la importación y exportación de aplicaciones, que adaptándolo al alas PACSManager se traducirá en registrar y cargar módulos del sistema alas PACS.

También se utilizarán las tecnologías existentes en el mundo para el desarrollo de soluciones de este tipo, explotando las funcionalidades que brinda cada una de estas.

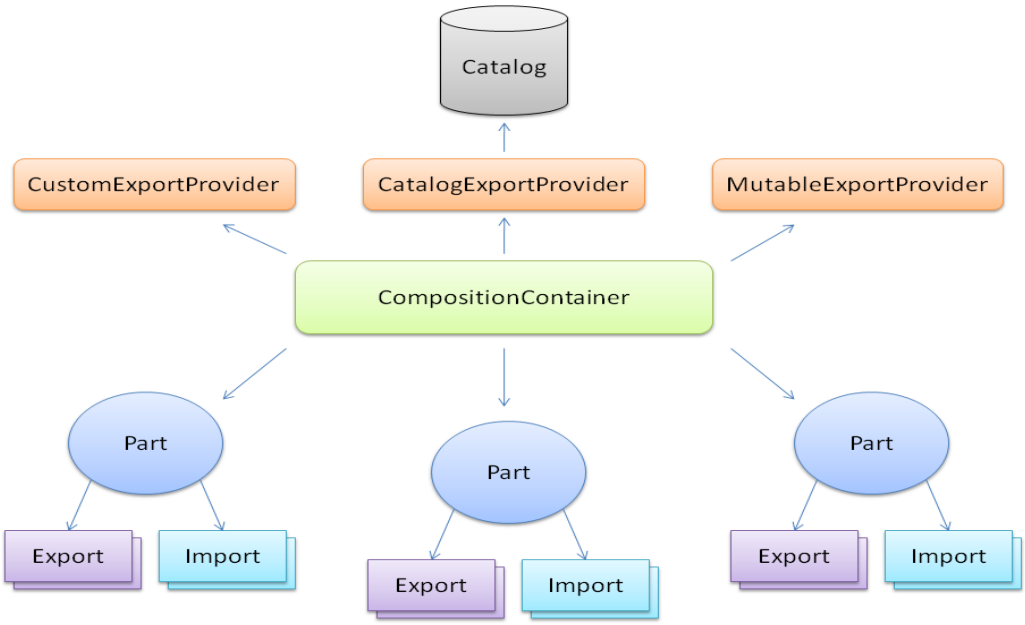
1.2 Tecnología a Evaluar.

Para el desarrollo de este tipo de servicio informático se evaluaron distintas tecnologías con el fin de definir con cuales se implementará el mismo. Estas presentan características que permiten a las soluciones integrales ser extensibles, además de brindar comunicación entre sus módulos. A continuación se exponen las características de las tecnologías evaluadas.

1.2.1 Managed Extensibility Framwork (MEF) *[6]*

Es una librería .NET que habilita escenarios de re-utilización de aplicaciones y componentes, haciendo posible que aplicaciones .NET compiladas de forma estática puedan ser compuestas de forma dinámica. MEF está pensado para escenarios en los que se está implementando aplicaciones y frameworks extensibles o bien extensiones para aplicaciones.

Esquema funcional de MEF:



En lugar de este registro explícito de componentes disponibles, MEF proporciona una manera de detectarlos implícitamente, a través de la composición. Un componente MEF, denominado un elemento, especifica mediante declaración tanto sus dependencias (conocidas como importaciones) como qué capacidades (conocidas como exportaciones) están disponibles. Cuando se crea un elemento, el motor de composición de MEF cubre sus importaciones con lo que está disponible en otros elementos

Dado que los elementos MEF especifican sus capacidades mediante declaración, son reconocibles en tiempo de ejecución, lo que significa que una aplicación puede utilizar elementos sin referencias incluidas en el código o archivos de configuración frágiles. MEF permite a las aplicaciones detectar y examinar elementos por sus metadatos, sin crear instancias ni cargar sus ensamblados. Por consiguiente, no hay necesidad de especificar meticulosamente qué extensiones se deben cargar y cuándo.

Además de las exportaciones proporcionadas, un elemento puede especificar las importaciones, que serán completadas por otros elementos. Esto hace que la comunicación entre los elementos sea posible además

de sencilla y permite una buena factorización del código. Por ejemplo, los servicios comunes a muchos componentes se pueden tener en cuenta en un elemento independiente, para modificarlos o reemplazarlos con facilidad.

Dado que el modelo MEF no requiere dependencias lógicas en un ensamblado de aplicación determinado, permite reutilizar las extensiones entre una aplicación y otra. Esto también facilita el desarrollo de un agente de prueba, independiente de la aplicación, para probar los componentes de extensión.

Una aplicación extensible escrita utilizando MEF declara una importación que puede ser completada por componentes de extensión y también puede declarar exportaciones para exponer servicios de aplicación a extensiones. Cada componente de extensión declara una exportación y también puede declarar importaciones. De esta manera, los componentes de extensión son automáticamente extensibles.

1.2.2 Managed Add-in Framework (MAF) *[7]*

Es un conjunto de ensamblados que forman parte de .NET Framework v3.5. Ellos ofrecen una plataforma para los desarrolladores construir aplicaciones de extensibilidad en su cliente por addins de habilitación (también conocido como plug-ins) para ser escrito por un huésped. MAF ofrecen características extendidas o servicios de una aplicación host. .NET Framework, proporciona un modelo de programación que los desarrolladores pueden utilizar para desarrollar complementos y activar en su aplicación host. El modelo logra esto mediante la construcción de una canalización de comunicación entre el host y el complemento. El modelo se implementa mediante el uso de los tipos en los espacios de nombres System.AddIn, System.AddIn.Hosting, System.AddIn.Pipeline y System.AddIn.Contract.

El modelo de complementos se compone de una serie de segmentos que componen el complemento de la tubería (también conocida como la canalización de la comunicación), que es responsable de todas las comunicaciones entre el complemento y el host. El oleoducto es un modelo de comunicación simétrica de los segmentos que el intercambio de datos entre un complemento y su host. El desarrollo de estos segmentos entre el host y el complemento proporciona las capas de abstracción que requiere el apoyo de versiones y el aislamiento del complemento.

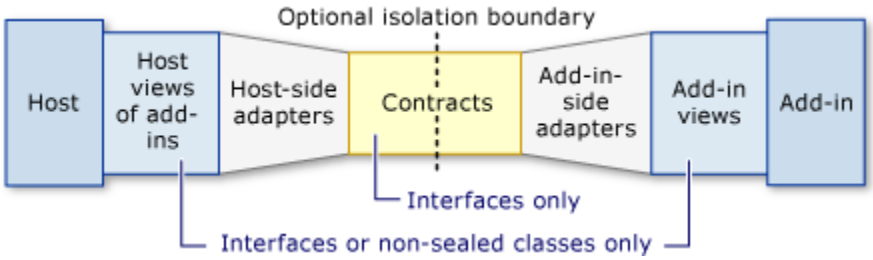


Fig. : Modelo de tuberías de Add-in.

Los montajes de estos segmentos no están obligados a estar en el mismo dominio de aplicación. Puede cargar un plugins en su propio dominio de aplicación nuevo, en un dominio de aplicación existente, o incluso en el dominio de aplicación del host. Puede cargar múltiples plugins en el mismo dominio de aplicación, que permite estos compartir recursos y contextos de seguridad.

El modelo de soporte de Add-ins recomienda un límite opcional entre el host y el plugins, que se llama el límite de aislamiento (también conocida como límite remoto). Este límite puede ser un dominio de aplicación o del límite del proceso.

El segmento de contrato en el centro de la tubería se carga en tanto dominio de la aplicación del host y el plugins de dominio de aplicación. El contrato define los métodos virtuales que el host y el plugins en el uso de tipos de cambio entre sí.

Para pasar a través del límite de aislamiento, los tipos deben ser los contratos o serializables. Los tipos que no son contratos o serializables deben convertirse a contratos por parte de los segmentos del adaptador en la tubería. Los segmentos de vista de la tubería son clases abstractas base o interfaces que proporcionan el host y el complemento con el fin de los métodos que comparten, tal como se define en el contrato.

El modelo de complementos permite a los hosts y complementos a la versión independiente. Como resultado, el modelo de complementos permite a los siguientes escenarios:

Creación de un adaptador que permite a un host para utilizar un complemento creado para una versión anterior de la máquina.

Creación de un adaptador que permite a un host para utilizar un complemento creado para una versión posterior de la máquina.

Creación de un adaptador que permite a un host para utilizar los complementos construidos para un host diferente.

1.2.3 Remoting/IPC *[8]*

NET Remoting permite crear fácilmente aplicaciones ampliamente distribuidas, tanto si los componentes de las aplicaciones están todos en un equipo como si están repartidos por el mundo. Se pueden crear aplicaciones de cliente que utilicen objetos en otros procesos del mismo equipo o en cualquier otro equipo disponible en la red. También se puede utilizar .NET Remoting para comunicarse con otros dominios de aplicación en el mismo proceso.

.NET Remoting proporciona un enfoque abstracto en la comunicación entre procesos que separa el objeto utilizado de forma remota de un dominio de aplicación de cliente o servidor específico y de un mecanismo específico de comunicación. Por lo tanto, se trata de un sistema flexible y fácilmente personalizable. Se puede reemplazar un protocolo de comunicación con otro o un formato de serialización con otro sin tener que recompilar el cliente ni el servidor. Además, el sistema de interacción remota no presupone ningún modelo de aplicación en particular. Se puede comunicar desde una aplicación Web, una aplicación de consola, un servicio de Windows, desde casi cualquier aplicación que se desee utilizar.

Los servidores de interacción remota también pueden ser cualquier tipo de dominio de aplicación. Cualquier aplicación puede albergar objetos de interacción remota y proporcionar sus servicios a cualquier cliente en su equipo o red.

La ventaja del sistema de interacción remota es su capacidad para permitir la comunicación entre objetos pertenecientes a dominios de aplicación o a procesos distintos mediante diferentes protocolos de transporte, formatos de serialización, esquemas de duración de objetos y modos de creación de objetos. Además, la interacción remota permite intervenir en prácticamente todas las fases del proceso de comunicación, sea cual sea la razón.

Tanto si implementó una serie de aplicaciones distribuidas como si sólo desea mover componentes a otros equipos para aumentar la escalabilidad de su programa, le resultará más fácil si considera el sistema de interacción remota como un sistema genérico de comunicación entre procesos con algunas implementaciones predeterminadas capaces de controlar fácilmente la mayoría de los escenarios posibles.

La arquitectura de simplificada de interacción remota está basada uso de referencias a objetos para la comunicación entre objetos de servidor y clientes es la esencia de la interacción remota. No obstante, la arquitectura de interacción remota proporciona al programador un procedimiento aún más sencillo. Si configura correctamente el cliente, sólo tiene que crear una nueva instancia del objeto remoto mediante new (o la función de creación de instancias del lenguaje de programación administrado que utilice). Su cliente recibe una referencia al objeto de servidor, lo que le permite llamar a sus métodos como si el objeto estuviera en su proceso en lugar de estar ejecutándose en otro equipo.

El sistema de interacción remota utiliza objetos proxy para dar la impresión de que el objeto del servidor se encuentra en el proceso del cliente. Los objetos proxy son objetos complementarios, que se presentan como si fueran otro objeto. Cuando un cliente crea una instancia del tipo remoto, la infraestructura de interacción remota crea un objeto proxy que, para su cliente, tiene exactamente la misma apariencia que el tipo remoto. Su cliente llama a un método en ese objeto proxy y el sistema de interacción remota recibe la llamada, la dirige hacia el proceso del servidor, invoca al objeto de servidor y envía el valor devuelto al objeto proxy del cliente, que a su vez devuelve el resultado al cliente.

Las llamadas remotas deben ser transmitidas de alguna forma entre el cliente y el proceso del servidor. Si crea un sistema de interacción remota por su cuenta, podría empezar aprendiendo programación de redes y una amplia gama de protocolos y especificaciones de formatos de serialización. En el sistema .NET Remoting, la combinación de tecnologías subyacentes necesarias para abrir una conexión de red y utilizar un determinado protocolo para enviar los bytes a la aplicación receptora se representa como un canal de transporte.

Un canal es un tipo que toma una secuencia de datos, crea un paquete según un determinado protocolo de red y lo envía a otro equipo. Algunos canales sólo pueden recibir información, otros sólo pueden enviarla y otros, como las clases predeterminadas TcpChannel y HttpChannel, se pueden utilizar en ambos sentidos.

Aunque el proceso del servidor conoce perfectamente todos los tipos, el cliente sólo sabe que necesita una referencia a un objeto de otro dominio de aplicación, puede que de otro equipo. Desde el exterior del dominio de aplicación de servidor, una dirección URL ubica el objeto. Las direcciones URL que representan

tipos únicos para el mundo exterior son direcciones URL de activación, que garantizan que su llamada remota va dirigida al tipo apropiado. Para obtener más información, vea Direcciones URL de activación. [8]

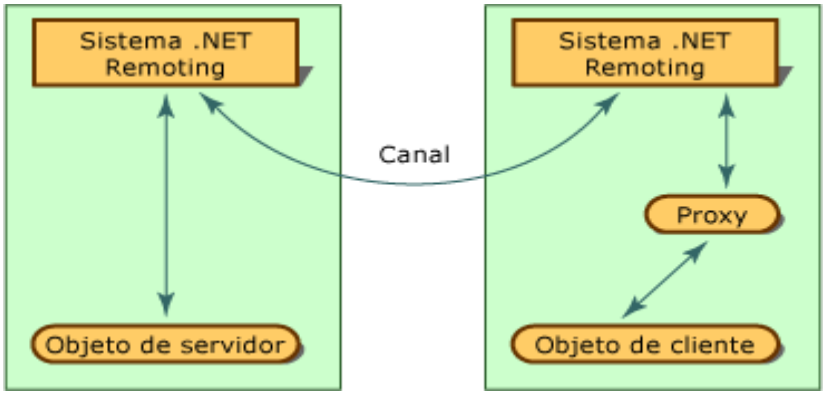


fig. Diseño de un sistema de interacción remota

1.2.4 D-Bus

Es esencialmente la comunicación entre procesos (IPC) de una aplicación. Este está diseñado para aplicaciones de escritorio y las comunicaciones del sistema operativo de baja latencia y bajo costo operativo, diseñado para ser pequeño y eficiente, con el fin de minimizar el tiempo de ida y vuelta de transmisión. Además, utiliza protocolo binario, no en texto, por lo que descarta la secuencia de proceso que consume tiempo, D-Bus se utiliza para software libre y aplicaciones de Linux. [9]

1.2.5 Windows Communication Foundation (WCF)

Es un marco de trabajo para la creación de aplicaciones orientadas a servicios. Con WCF, es posible enviar datos como mensajes asincrónicos de un extremo de servicio a otro. Un extremo de servicio puede formar parte de un servicio disponible continuamente hospedado por Internet InformationServices (IIS), o puede ser un servicio hospedado en una aplicación. Un extremo puede ser un cliente de un servicio que solicita datos de un extremo de servicio. Los mensajes pueden ser tan simples como un carácter o una palabra que se envía como XML, o tan complejos como una secuencia de datos binarios.

1.2.5.1 Características de WCF: *[10]*

- ✓ **Orientación a servicios.**

Como consecuencia del uso de los estándares de WS, WCF le permite crear aplicaciones *orientadas a servicios*. SOA, la arquitectura orientada a servicios es el uso de servicios web para enviar y recibir datos. Los servicios tienen la ventaja general de estar débilmente acoplados entre una aplicación y otra en lugar de incluidos en el código. Una relación de acoplamiento débil implica que cualquier cliente creado en cualquier plataforma puede conectar con cualquier servicio siempre y cuando se cumplan los contratos esenciales.

- ✓ **Interoperabilidad**

WCF implementa estándares modernos de la industria para la interoperabilidad de servicios web. Para obtener más información sobre los estándares admitidos.

- ✓ **Varios modelos de mensajes**

Los mensajes se intercambian mediante uno de los distintos modelos. El más común es el de solicitud/respuesta, en que un extremo solicita datos de otro extremo y el otro extremo responde. Existen otros modelos, como un mensaje unidireccional, en que un único extremo envía un mensaje sin esperar ninguna respuesta. Un modelo más complejo es el modelo de intercambio de mensajes dúplex, en donde dos extremos establecen una conexión y se envían datos entre sí, como ocurre con un programa de mensajería instantánea. Para obtener más información sobre cómo implementar los distintos modelos de intercambio de mensajes utilizando WCF.

- ✓ **Metadatos de servicios**

WCF admite la publicación de metadatos de servicios utilizando los formatos especificados en los estándares de la industria, como WSDL, Esquemas XML y WS-Policy. Estos metadatos pueden utilizarse para generar y configurar automáticamente clientes para el acceso a los servicios de WCF. Los metadatos se pueden publicar sobre HTTP y HTTPS, o utilizando el estándar Intercambio de metadatos de servicios web. Para obtener más información.

- ✓ **Contratos de datos**

Dado que WCF se basa en .NET Framework, también incluye métodos con código sencillo para proporcionar los contratos que desea aplicar. Uno de los tipos de contrato universales es el contrato de datos. Básicamente, mientras se escribe el código del servicio utilizando Visual C# o Visual Basic, la forma más sencilla de controlar los datos consiste en crear clases que representan una entidad de datos con propiedades que pertenecen a la misma. WCF incluye un completo sistema para trabajar con los datos de esta manera tan sencilla. Cuando se han creado las clases que representan los datos, el servicio genera automáticamente los metadatos que permiten a los clientes ajustarse a los tipos de datos que se han diseñado.

- ✓ **Seguridad**

Es posible cifrar los mensajes para proteger la privacidad, así como obligar a los usuarios a que se autentiquen antes de permitirles recibir mensajes. La seguridad puede implementarse utilizando estándares conocidos como SSL o WS-Secure Conversation.

- ✓ **Varios transportes y codificaciones**

Los mensajes pueden enviarse con cualquiera de los protocolos y codificaciones integrados. La combinación más frecuente de protocolo y codificación consiste en enviar mensajes SOAP codificados de texto utilizando el Protocolo de transferencia de hipertexto (HTTP) usado en World Wide Web. WCF también le permite enviar mensajes sobre TCP, canalizaciones con nombre o MSMQ. Estos mensajes pueden codificarse como texto o utilizando un formato binario optimizado. Los datos binarios pueden enviarse de manera eficaz utilizando el estándar MTOM. Si ninguno de los transportes o codificaciones proporcionados satisface sus necesidades, puede crear uno personalizado.

- ✓ **Mensajes confiables y en cola**

WCF admite el intercambio de mensajes confiable utilizando sesiones confiables implementadas sobre WS-Reliable Messaging y utilizando MSMQ.

- ✓ **Mensajes duraderos**

Un mensaje duradero es aquel que nunca se pierde debido a una interrupción de la comunicación. Los mensajes que forman parte de un modelo de mensajes duraderos siempre se guardan en una base de datos. Si se produce una interrupción, la base de datos le permite reanudar el intercambio de mensajes

cuando se restablezca la conexión. También puede crear un mensaje duradero utilizando Windows Workflow Foundation (WF).

- ✓ **Transacciones**

WCF también admite las transacciones que usan uno de los tres modelos de transacción: las transacciones WS-Atomic, las API del espacio de nombres System.Transactions y Microsoft DTC (Coordinador de transacciones distribuidas).

- ✓ **Compatibilidad con AJAX y REST**

REST es un ejemplo de una tecnología de web 2.0 en evolución. WCF se puede configurar para procesar datos XML "sin formato" que no se ajustan en un sobre SOAP. WCF también se puede extender para admitir formatos XML concretos, como ATOM (un estándar popular de RSS) e incluso formatos que no sean XML, como JavaScript ObjectNotation (JSON).

- ✓ **Extensibilidad**

La arquitectura de WCF tiene varios puntos de extensibilidad. Si se necesita una función adicional, existen una serie de puntos que le permiten personalizar el comportamiento de un servicio.

1.2.5.2 Integración de WCF con otras tecnologías de Microsoft:

WCF es una plataforma flexible. Debido a esta flexibilidad extrema, WCF también se usa en varios otros productos Microsoft. Si comprende los fundamentos de WCF, tendrá una ventaja inmediata si también utiliza cualquiera de estos productos.

La primera tecnología en adaptarse a WCF fue Windows Workflow Foundation (WF). Los flujos de trabajo simplifican el desarrollo de aplicaciones encapsulando los pasos del flujo de trabajo como “actividades”. En la primera versión de Windows Workflow Foundation, un desarrollador tenía que crear un host para el flujo de trabajo. La versión siguiente de Windows Workflow Foundation se integró con WCF. Esto permitió hospedar cualquier flujo de trabajo fácilmente en un servicio de WCF; puede hacer esto si elige automáticamente el tipo de proyecto WF/WCF en Visual Studio 2010.

Microsoft BizTalk Server R2 también utiliza WCF como tecnología de comunicaciones. BizTalk está diseñado para recibir y transformar datos de un formato normalizado en otro. Los mensajes deben entregarse en su cuadro de mensajes central, donde es posible transformar el mensaje utilizando una asignación estricta o mediante una de las características de BizTalk, como su motor de flujo de trabajo. BizTalk ahora puede utilizar el adaptador de línea de negocio (LOB, Line Of Business) de WCF para entregar mensajes en el cuadro de mensajes.

Microsoft Silverlight es una plataforma para la creación de sofisticadas aplicaciones web interoperables que permiten a los desarrolladores crear sitios Web con uso intensivo de contenidos multimedia (como la transmisión de vídeo por secuencias). A partir de la versión 2, Silverlight incorpora WCF como tecnología de comunicaciones para conectar las aplicaciones Silverlight con los extremos de WCF.

Microsoft .NET Services es una iniciativa de computación en nube (cloudcomputing) que utiliza WCF para la creación de aplicaciones habilitadas para Internet. Utilice .NET Services para crear servicios WCF que funcionan a través de límites de confianza.

El servidor de aplicaciones características de hospedaje de Windows Server AppFabric se ha diseñado específicamente para implementar y administrar aplicaciones que utilizan WCF para las comunicaciones. Características de hospedaje incluye sofisticadas opciones de configuración y herramientas diseñadas específicamente para las aplicaciones habilitadas para WCF. [10]

1.2.6 Análisis crítico de las tecnologías.

Para el desarrollo del servicio alas PACSManager se utilizará las tecnologías WCF y MEF. A continuación se muestran las características que permitieron la elección de dichas tecnologías:

La publicación de servicios de Windows Communication Foundation (WCF) ayuda a pasar del entorno de desarrollo preliminar que proporciona el host de servicio de WCF y el cliente de prueba de WCF a la implementación real de la aplicación en un entorno producción con fines de prueba. Antes de confirmar un plan de implementación final, puede usar la publicación de servicios de Windows Communication Foundation (WCF) para comprobar que el servicio de WCF funciona correctamente y está listo para su

publicación, esto permitirá al alas PACSManager ser probado en sus distintas fases de desarrollo posibilitando la incorporación y publicación de servicios antes de obtener una aplicación final.

También puede decidir implementar las bibliotecas de servicios de WCF en varias ubicaciones de destino para comprobarlas. WCF soporta SOAP (Simple Object Access Protocol), protocolo que se asemeja al XML el cual tiene la particularidad de ser configurado con atributos desde las mismas clases que se quieren pasar en la comunicación. Este protocolo permite encriptar y asegurar información a través del internet, se puede hostear un servicio sin que este en el IIS o en los servicios administrados de Windows, soporta patrones de intercambio de mensajes como solicitud/respuesta, unidireccional, y duplex; incluso soporta Peer networking donde los clientes no necesitan de un mecanismo central para establecer comunicación así como las redes mesh.

La implementación de este protocolo proporciona al alas PACSManager seguridad en la comunicación de los módulos del alas PACS. WCF permite describir, publicar, implementar y consumir servicios, no solo con la interoperabilidad de los Web Services entre plataformas servidor y cliente, sino también utilizando diferentes plataformas de transporte de forma transparente al resto de la arquitectura. Con esta característica la aplicación alas PACSManager puede enviar y recibir mensajes asincrónicos sin la necesidad de modificar la arquitectura de la solución alas PACS.

También se utilizará la tecnología MEF ya que:

- ✓ Es una librería .NET que habilita escenarios de re-utilización de aplicaciones y componentes, haciendo posible que aplicaciones .NET compiladas de forma estática puedan ser compuestas de forma dinámica.
- ✓ Está pensada para escenarios en los que se está implementando aplicaciones y frameworks extensibles o bien extensiones para aplicaciones.
- ✓ Permite a las aplicaciones detectar y examinar elementos por sus metadatos, sin crear instancias ni cargar sus ensamblados. Por consiguiente, no hay necesidad de especificar meticulosamente qué extensiones se deben cargar y cuándo.
- ✓ Dado que el modelo MEF no requiere dependencias lógicas en un ensamblado de aplicación determinado, permite reutilizar las extensiones entre una aplicación y otra. Esto también facilita el

desarrollo de un agente de prueba, independiente de la aplicación, para probar los componentes de extensión.

El uso de MEF brinda extensibilidad a la solución alas PACS así como la carga de módulos de forma más rápida debido a que este no necesita crear instancias ni cargar ensamblados para detectar los módulos de la solución alas PACS. Además MEF permite la reutilización de componentes lo que posibilita al alas PACSManager que el trabajo con las actualizaciones de los módulos del alas PACS sea más fácil.

Otras tecnologías como MAF y NET Remoting no serán utilizadas debido a que estas se usan principalmente para diferentes dominios de aplicación, lo cual no se necesita, por ser el alas PACSManager utilizado en cada una de las estaciones de visualización. En cuanto a D-bus, se descarta por ser utilizado para aplicaciones de Linux.

METODOLOGÍA Y HERRAMIENTAS:

1.3 Metodología de Desarrollo de Software:

Atendiendo a las características de las herramientas y metodologías existentes y a su compatibilidad con la aplicación se utilizará:

RUP [11]

Las siglas RUP en inglés significa Rational Unified Process (Proceso Unificado de Desarrollo) es un producto del proceso de ingeniería de software que proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización del desarrollo. Su meta es asegurar la producción del software de alta calidad que resuelve las necesidades de los usuarios dentro de un presupuesto y tiempo establecido. RUP es dirigido por casos de uso, centrado en arquitectura e iterativo incremental; además permite clasificar el desarrollo de software en faces (Comienzo o Inicio, Elaboración, Construcción, Transición) permitiendo comprobar los objetivos del desarrollo de software para cada una de las faces.

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo, los cuales son Modelamiento del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba o Testeo, Instalación, Administración de Configuración y Cambios y Ambiente. Una particularidad de esta metodología

es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.2 Lenguaje de Modelado:

1.3.1UML 2.0 *[12]*

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas.

Ventajas

- ✓ Es un lenguaje conocido
- ✓ Estándar
- ✓ Fácil de aprender.

Desventajas

- ✓ No ha sido diseñado para modelar procesos de negocios
- ✓ Implica un enfoque orientado a objetos
- ✓ UML no tiene todavía una semántica formal.

1.4 Herramientas:

1.4.1 Enterprise Architect 7.5

Enterprise Architect es una herramienta de multi-usuarios, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Además, ofrece salida de documentación flexible y de alta calidad. Su interfaz intuitiva y de alto rendimiento con una vista del proyecto como un "explorador", además posee un entorno fácil de usar, rápido y flexible

Las bases de Enterprise Architect usan Perfiles UML para extender el dominio de modelado, mientras que la validación del modelo asegura integridad. Combina los procesos de negocio, información y flujos de trabajo en un modelo usando extensiones gratuitas para la notación de modelado de procesos del negocio (BPMN).

Provee trazabilidad completa desde el análisis de requerimientos hasta los artefactos de análisis y diseño, a través de la implementación y el despliegue. Combinados con la ubicación de recursos y tareas incorporados, los equipos de Administradores de Proyectos y Calidad están equipados con la información que ellos necesitan para ayudarles a entregar proyectos en tiempo.

Enterprise Architect permite elegir los modelos que se quieren utilizar. Una vez elegidos crea una estructura con elementos de ejemplo que pueden guiar muy bien.

Por ser el Enterprise Arquitect fácil de manejar, con una interfaz amigable e intuitiva, por usar perfiles UML para extender el dominio de modelado y por ofrecer salida de documentación flexible y de alta calidad, se decide utilizar el Enterprise Architect como herramienta de modelado. [13]

1.4.2 Visual Studio 2010:

Como entorno de desarrollo integrado (IDE) se ha seleccionado al Microsoft Visual Studio. El mismo soporta varios lenguajes de programación como son Visual C++, Visual J#, Visual Basic y Visual C#. Visual Studio provee efectivas herramientas que permiten administrar proyectos y mantener el código fuente, contiene un editor el cual soporta resaltado de sintaxis y completamiento de código. A su vez este IDE permite llevar a cabo tareas de modelado, codificado, prueba o depuración, sin salir de su entorno, brindando alta eficiencia y productividad.

Esta versión incluye al Microsoft .NET Framework 4, el cual es el modelo de programación completo y coherente de Microsoft para compilar aplicaciones que ofrezcan una sensacional experiencia visual del usuario, comunicación perfecta y segura, y la capacidad de modelar una amplia gama de procesos empresariales. [14]

Luego del estudio del arte no se encontró ningún sistema de este tipo a nivel nacional. En el ámbito internacional, Microsoft BizTalk Server, no satisface las necesidades de integración desacoplada, configuración y comunicación entre los módulos del alas PACS en una estación de visualización. Además, dentro de las tecnologías para la implementación de este tipo de aplicación, se seleccionó; para el desarrollo del alas PACSManager, MEF (Managed Extensibility Framework) y WCF (Windows Communication Foundation).

Por sus características y atendiendo a la compatibilidad con la aplicación alas PACSManager, se escogió como metodología de desarrollo RUP, como lenguaje de modelado se utilizará UML. Para lograr un mejor entendimiento en el modelo del mismo se utilizará Enterprise Architect 7.5, así como para su desarrollo se escogió Visual Studio 2010.

CAPÍTULO 2. Características del Sistema

En este capítulo se realiza un análisis a partir de los resultados arrojados de la investigación previa donde es necesaria la identificación de los conceptos asociados al Modelo de Dominio, para definir los requisitos funcionales y no funcionales, además de un prototipo de interfaz externa. Se definirá el modelo de Análisis y diseño del sistema. Y finalmente quedarán definidos los procesos que se automatizarán en la aplicación.

2.1. Objeto de automatización

Implementar un servicio que logre la integración, configuración y manejos de componentes en una estación de visualización.

2.2. Propuesta del sistema

Se propone la implementación de un servicio que gestione la interacción y configuración de instancias de componentes en una estación de visualización a través de mensajes y reglas.

2.3. Modelo de Dominio

Modelo de Dominio o Modelo Conceptual, es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Define un modelo de clases común para todos los implicados en el desarrollo, representadas en objetos del dominio, sirve como interlocutor entre clientes y desarrolladores. El propósito fundamental de este modelo es generar una terminología común y sentar las bases del entendimiento del desarrollo y no para definir el sistema completo. Cualquiera sea la solución de casos de uso que se haya elegido, los conceptos e ideas propias del dominio del problema son las mismas. Un mismo modelo de dominio contempla cualquiera de las soluciones analizadas. El modelo de dominio es global, es decir, se realiza para todos los casos de uso y no para uno en particular. [15]

2.4. Conceptos identificados:

Manager: Encierra todos los eventos y componentes capaces de gestionar las instancias y mensajes a través de reglas

Herramienta de configuración: Configura de forma centralizada, los componentes que están en el manager desde fuera del entorno del mismo.

Servicio: Es el encargado de servir de interfaz entre los componentes recibiendo mensajes de petición y basado en sus reglas actuar en consecuencia del dominio.

Reglas: Conjunto de políticas que están implícitas en el sistema o servicio y que condicionan el flujo de eventos en este entorno.

Instancia: Componente que está en una estación de visualización determinada.

Gestor de mensajes: Se encarga de filtrar, o sea, aceptar o rechazar los mensajes provenientes de una instancia.

Gestor de instancias: Se encarga del manejo de instancias por medio de mensajes en cada estación de trabajo.

Visor: Componente del alas PACS que permite visualizar y a la vez brinda numerosas opciones para el análisis de las distintas imágenes DICOM ayudando a dar una mayor precisión al especialista en el diagnóstico del paciente en cuestión.

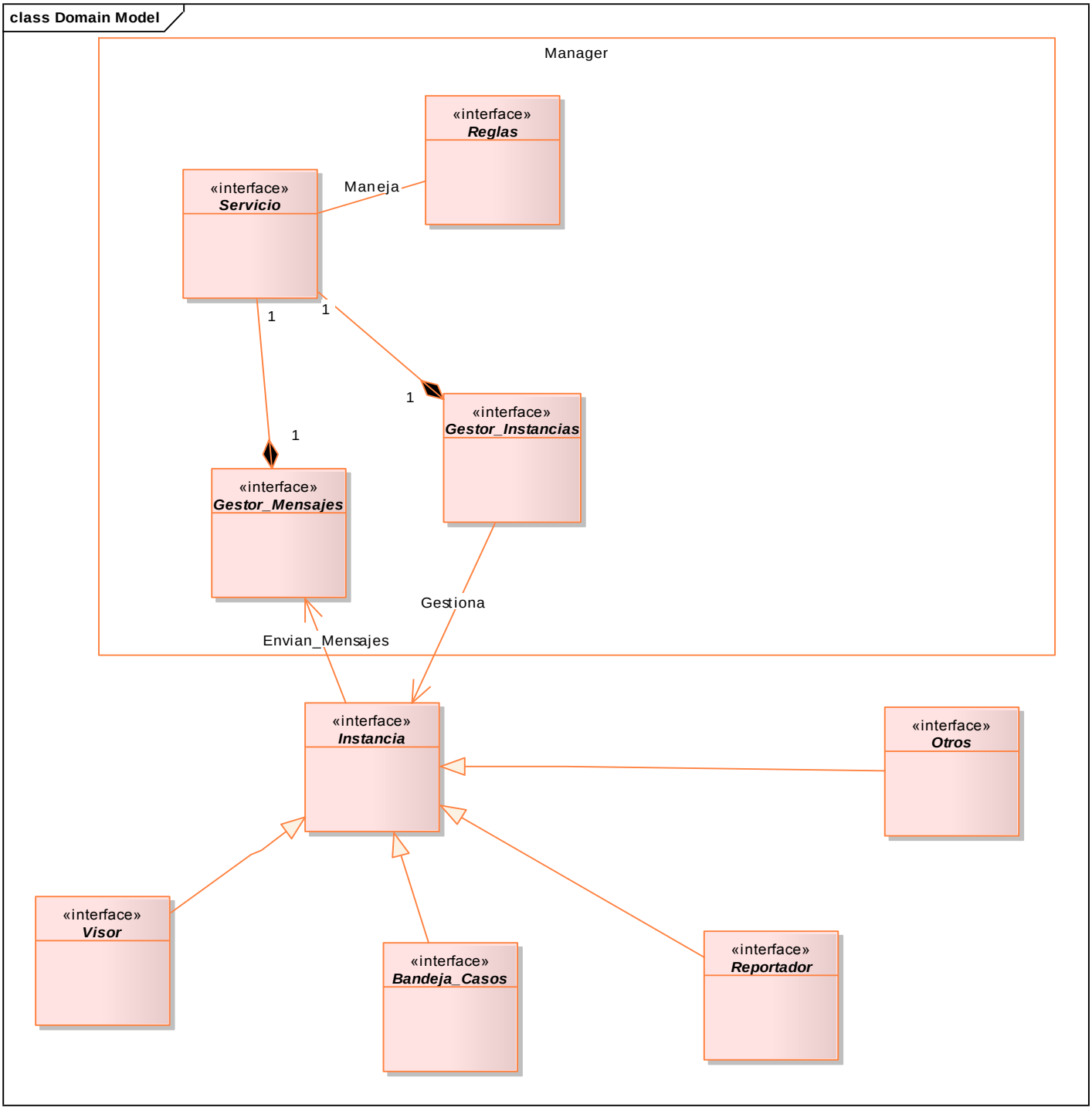
Bandeja de Casos: Está contenida en el visor y permite la búsqueda de imágenes DICOM por distintos criterios como son estudio, paciente, sexo, fecha entre otros; además de la obtención de dichas imágenes.

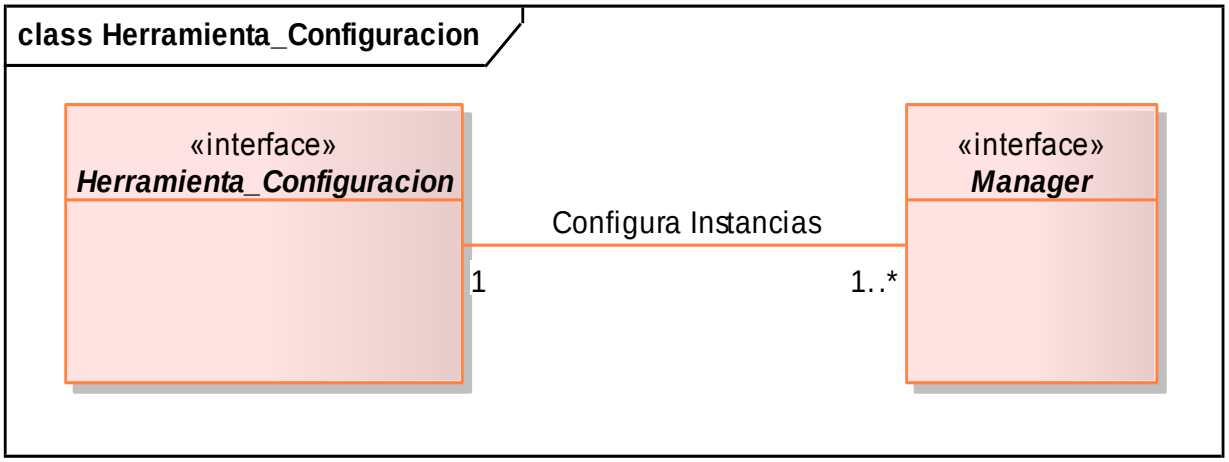
Reportador: Se encarga de enviar al RIS los diagnósticos o reportes de cada imagen.

Componente: Bandeja de Casos, Reportador, Visor y Otros.

Otros: Otras instancias que puedan surgir a medida que vaya avanzando el desarrollo de la aplicación. Un ejemplo de una de ellas puede ser un quemador de imágenes.

A raíz de la descripción anterior, el modelo de dominio se estructuró de la siguiente forma:


class Domain Model
Manager
«interface»
Reglas
«interface»
Servicio
Maneja
1
1
1
1
«interface»
Gestor_Instancias
«interface»
Gestor_Mensajes
Gestiona
Envian_Mensajes
«interface»
Instancia
«interface»
Otros
«interface»
Visor
«interface»
Bandeja_Casos
«interface»
Reportador



2.5. Especificación de los requisitos software.

Este servicio está concebido como parte del software alas PACS para el manejo de componentes de una estación de visualización asociados al mismo como son: la Bandeja de Casos, El Reportador, El Visor y cualquier otro modulo que se le agregue a dicha solución.

2.5.1. Requisitos Funcionales.

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir, especifican acciones que el sistema debe ser capaz de realizar. Ellos deben ser comprensibles por los clientes, usuarios y desarrolladores, deben tener una sola interpretación y estar definidos en forma medible y verificable. Para el presente sistema los mismos están agrupados en cinco paquetes fundamentales.

a) **Gestionar Módulo:** Especifica las acciones que debe permitir el servicio sobre los componentes de una estación de visualización.
b) **Gestionar Mensajes de módulos:** Específica cómo es la comunicación entre componentes en una estación de visualización.
c) **Gestionar Instancia:** Especifica los tipos de peticiones a pedir otras instancias de componentes de una estación de visualización.

d) **Gestionar Actualización:** Especifica los procesos correspondientes a la actualización de los componentes de una estación de visualización.

e) **Gestionar Configuración:** Especifica los procesos configurables de los componentes de una estación de visualización.

A continuación se listan los requerimientos asociados a cada paquete de requisitos con una breve descripción para su mejor comprensión.

1. **RF 1 Gestionar de Módulos.**

Requisito:	Descripción:
RF1.1: Cargar módulos.	Carga los metadatos de los componentes de la estación de visualización.
RF1.2: Validar certificados de módulos.	Valida que los módulos cargados en una estación de visualización no sean corruptos.

2. **RF 2 Gestionar Mensajes de módulos.**

Requisito:	Descripción:
RF 2.1: Enviar Mensajes.	Realiza envió de mensajes a componentes de una estación de visualización.
RF2.2: Recibir Mensajes.	Recibe mensajes desde componentes de una estación de visualización.

3. **RF 3 Gestionar de Instancia.**

Requisito:	Descripción:
RF3.1: Registrar Instancia	Registra una instancia de componente de una estación de visualización cuando es iniciada.
RF3.2: Cerrar Instancia.	Borra del registro una instancia de componente de una estación de visualización cuando es

	cerrada.
RF3.3: Buscar Instancia.	Busca una instancia de componente en una estación de visualización.

4. **RF 4 Gestionar Actualización.**

Requisito:	Descripción:
RF4.1: Buscar actualización de módulos.	Busca si existe alguna actualización para algún componente de una estación de visualización.
RF4.2: Aplicar actualización de módulos.	Aplica la actualización encontrada al componente de la estación de visualización.

5. **RF 5 Gestionar Configuración.**

Requisito:	Descripción:
RF 5.1: Aplicar configuración.	Aplica los cambios de la configuración a una componente de una estación de visualización.

2.5.2. Requisitos no funcionales.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable [16]

A continuación se presentarán los requerimientos no funcionales definidos para el sistema. Estos están divididos por categorías y cada una tiene asociado un prefijo con la que será identificada: Rendimiento: RNR, Diseño e Implementación: RNDI, Interconexión: RNI, Seguridad: RNS, Seguridad: RNS, Legales: RNL.

a) Rendimiento:

- ✓ RNR 1 Las tareas de búsqueda, obtención, envío y recepción de mensajes, entre otras; deben realizarse de forma asincrónica.
- ✓ RNR 2 La creación e interacción entre instancias de componente deben ser realizada de forma rápida con un tiempo máximo de 40 milisegundos, así como la carga, validación y registro de instancias de módulos.

b) Extensibilidad:

- ✓ RNE 1: Agregar módulo dinámicamente.

c) Seguridad:

- ✓ RNS 1 Los componentes a cargar deben ser certificados como que son parte de la solución integral impidiendo la carga de componentes corruptos ni que el sistema colapse por sobrecarga de módulos.

d) Legales:

- ✓ RNL 1 El sistema y toda la documentación generada con el mismo, pertenecen al Centro de Informática Médica (CESIM).

e) Diseño e Implementación:

- ✓ RNDI C# como lenguaje de programación.
- ✓ RNDI NET Framework vs 4.0.

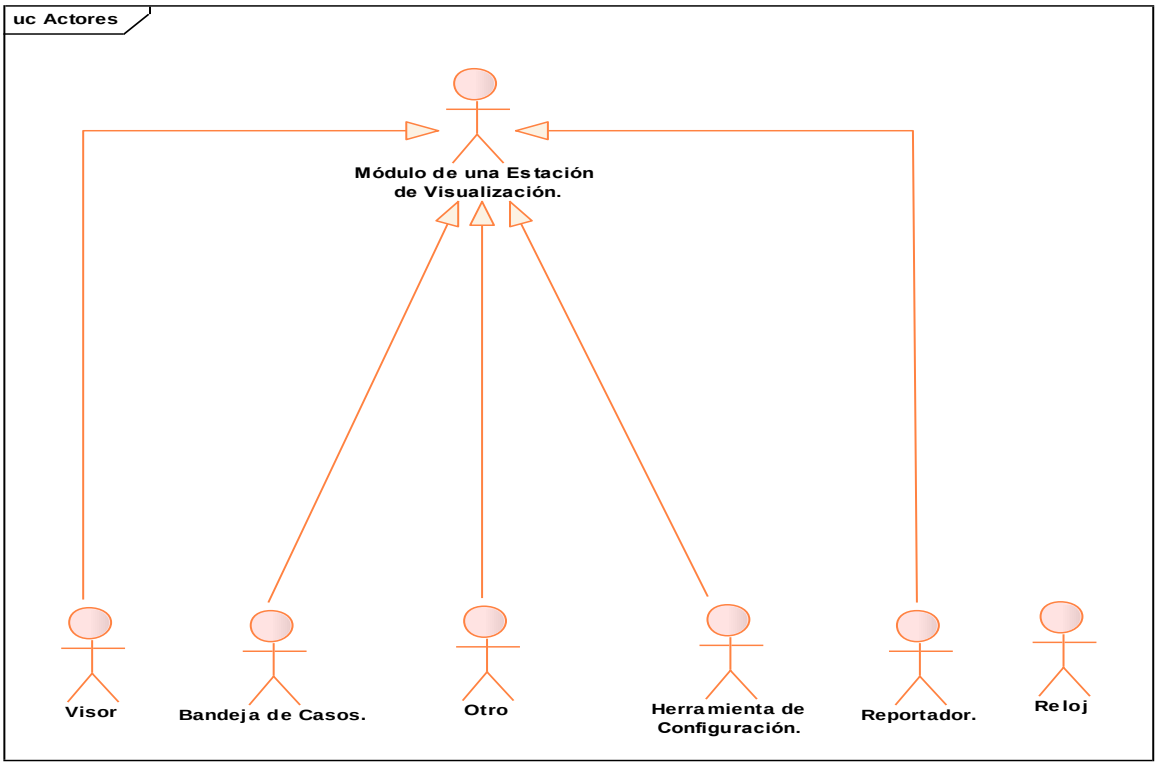
2.6. Definición de los Casos de Uso del Sistema.

Los diagramas de casos de uso especifican la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y otros sistemas. Se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona una respuesta a eventos que se producen en el mismo.

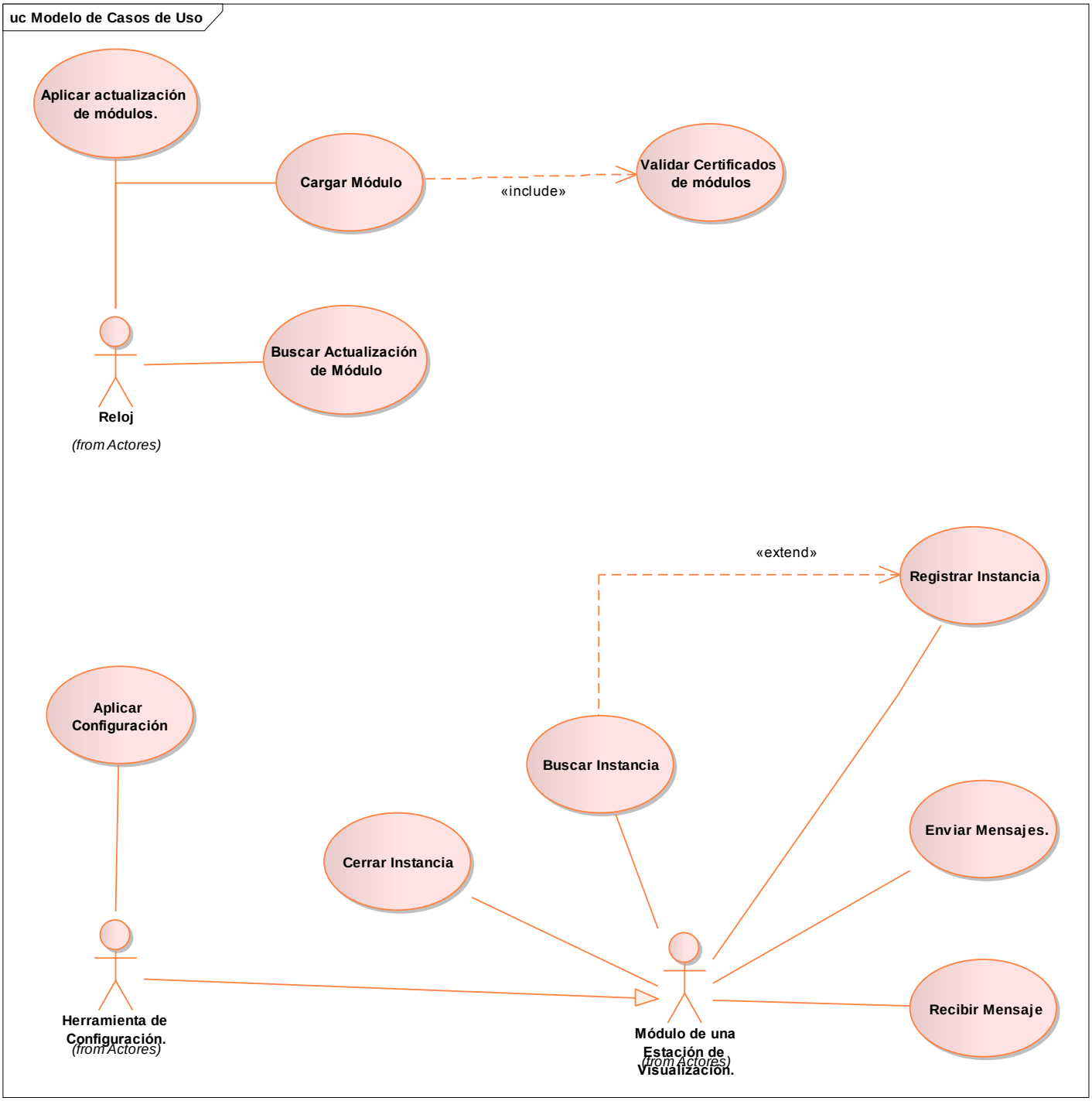
2.6.1. Actores del sistema.

Actores.	Justificación.
Visor.	Sistema para la visualización de imágenes médicas perteneciente a una estación de visualización.

Bandeja de Casos.	Sistema encargado de la obtención de estudios imagenológicos generados por los equipos de adquisición de imágenes.
Reportador.	Sistema encargado de confeccionar diagnósticos médicos, basado en el análisis de imágenes médicas.
Herramienta de Configuración.	Sistema encargado de la configuración de módulos de una estación de visualización (Reportador, Bandeja de Casos, Visor, Otro).
Otro.	Cualquier nuevo módulo que pueda ser integrado a una estación de visualización.
Reloj.	Inicia el proceso automático cargar módulos, validar certificados y buscar y aplicar actualización de módulos.
Módulos de Estación de Visualización.	Son los mismos módulos de la estación de visualización.



2.6.2. Diagrama de Casos de Uso.



2.6.3. Casos de Uso expandidos

Los casos de uso expandidos son muy útiles para alcanzar un conocimiento más profundo de los procesos y de los requerimientos. Constituyen un documento narrativo que describe la secuencia de eventos de un actor que utiliza el sistema para completar un proceso.

Caso de Uso No. 01. Cargar módulos.		
Objetivo	Descubrir los componentes existentes en la estación de visualización.	
Actores	Reloj,	
Resumen	Cargar metadatos de los componentes instalados en la estación de visualización	
Complejidad	Baja	
Prioridad	Crítico	
Referencias	RF1.1, RF1.2.	
Precondiciones		
Postcondiciones		
Flujo de eventos		
Flujo básico		
1. El sistema busca los componentes de una estación de visualización. 2. El sistema valida los certificados de cada uno de los componentes encontrados. 3. El sistema carga los metadatos de los componentes cuyo certificado coincidan con la solución alas PACS. 4. Termina el escenario.		
Flujos alternos		
Relaciones	CU incluidos	CU Validar Certificado de Módulos
	CU extendidos	No aplicable

Requisitos no funcionales	No Aplicable
Asuntos pendientes	No Aplicable

Caso de Uso No. 02. Validar Certificados de Módulos.		
Objetivo	Descubrir los componentes existentes en la estación de visualización.	
Actores	Reloj,	
Resumen	Validar que los componentes a cargar no sean corruptos.	
Complejidad	Baja	
Prioridad	Crítico	
Referencias	RF1.2.	
Precondiciones		
Postcondiciones		
Flujo de eventos		
Flujo básico		
1. El sistema valida los certificados de los componentes de una estación de visualización. El sistema valida los certificados de cada uno de los componentes encontrados. 2. Termina el escenario.		
Flujos alternos		
Relaciones	CU incluidos	No Aplicable
	CU extendidos	No Aplicable

Requisitos no funcionales	No Aplicable
Asuntos pendientes	No Aplicable

Caso de Uso No. 03. Enviar Mensajes.	
Objetivo	Responder peticiones de componentes de una estación de visualización.
Actores	Módulos de Estación de Visualización.
Resumen	El sistema de respuestas a peticiones en forma de mensajes recibidos desde otros compontes.
Complejidad	Baja
Prioridad	Crítico
Referencias	RF2.1
Precondiciones	Recibir una petición desde un componente de una estación de visualización.
Postcondiciones	
Flujo de eventos	
Flujo básico	
1. Un Módulo de Estación de Visualización envía una petición en forma de mensaje al servicio para otro componente 2. El sistema evalúa la petición y crea un mensaje para el componente al que fue envía la petición. 3. El sistema envía el mensaje al componente 4. Termina el escenario.	
Flujos alternos	

Relaciones	CU incluidos	CU Recibir Mensaje
	CU extendidos	No Aplicable
Requisitos no funcionales	No Aplicable	
Asuntos pendientes	No Aplicable	

Caso de Uso No. 04. Recibir Mensaje	
Objetivo	Escuchar peticiones de los componentes de una estación de visualización.
Actores	Módulos de Estación de Visualización.
Resumen	El sistema recibe peticiones en forma de mensajes desde compontes de una estación de visualización.
Complejidad	Baja
Prioridad	Crítico
Referencias	RF2.2
Precondiciones	
Postcondiciones	
Flujo de eventos	
Flujo básico	

1. Un Módulo de Estación de Visualización envía una petición al servicio
2. El sistema recibe la petición y la analiza.
3. El sistema crea una respuesta para la petición recibida.
4. Termina el escenario.

Flujos alternos		
Relaciones	CU incluidos	No Aplicable
	CU extendidos	No Aplicable
Requisitos no funcionales	No Aplicable	
Asuntos pendientes	No Aplicable	

Caso de Uso No. 05. Registrar Instancia.	
Objetivo	Registrar una instancia que sea inicializada.
Actores	Módulos de Estación de Visualización.
Resumen	El sistema registra una instancia que sea inicializada.
Complejidad	Baja
Prioridad	Importante
Referencias	RF 3.1, RF3.3.
Precondiciones	
Postcondiciones	
Flujo de eventos	
Flujo básico	

1. Un Módulo de Estación de Visualización envía un mensaje confirmando que es inicializada.
2. El sistema recibe la petición y la analiza.
3. El sistema registra que una instancia de dicho componente es inicializado. .
4. Termina el escenario.

Flujos alternos		
Relaciones	CU incluidos	No Aplicable
	CU extendidos	CU Buscar Instancia.
Requisitos no funcionales	No Aplicable	
Asuntos pendientes	No Aplicable	

Caso de Uso No. 06. Cerrar Instancia.	
Objetivo	Borrar una instancia del registro de instancias inicializada.
Actores	Módulos de Estación de Visualización.
Resumen	El sistema borra del registro una instancia inicializada.
Complejidad	Baja
Prioridad	Importante
Referencias	RF 3.2.
Precondiciones	La instancia tiene que estar inicializada.
Postcondiciones	
Flujo de eventos	
Flujo básico	

1. Un Módulo de Estación de Visualización envía un mensaje confirmando que es cerrado
2. El sistema borra del registro de instancias inicializada la instancia de dicho componente cerrada.
3. Termina el escenario.

Flujos alternos		
Relaciones	CU incluidos	No Aplicable
	CU extendidos	No Aplicable.
Requisitos no funcionales	No Aplicable	
Asuntos pendientes	No Aplicable	

Caso de Uso No. 07. Buscar Instancia.	
Objetivo	Buscar una instancia de componente.
Actores	Módulos de Estación de Visualización.
Resumen	El sistema busca una instancia para atender una petición enviada al mismo.
Complejidad	Baja
Prioridad	Importante
Referencias	RF 3.3.
Precondiciones	
Postcondiciones	
Flujo de eventos	
Flujo básico	
1. Un Módulo de Estación de Visualización envía un mensaje a una instancia de componente de una estación de visualización. 2. El sistema busca la instancia para la cual es enviada dicha petición. 3. Termina el escenario.	

Flujos alternos		
Relaciones	CU incluidos	No Aplicable
	CU extendidos	No Aplicable
Requisitos no funcionales	No Aplicable	
Asuntos pendientes	No Aplicable	

Caso de Uso No. 08. Buscar Actualización de Módulos.	
Objetivo	Encontrar nuevas actualizaciones para los componentes instalados.
Actores	Reloj
Resumen	El sistema busca las actualizaciones para los componentes instalados.
Complejidad	Baja
Prioridad	Auxiliar
Referencias	RF 4.1.
Precondiciones	
Postcondiciones	
Flujo de eventos	
Flujo básico	
1. El sistema busca nuevas actualizaciones para los componentes instalados en la estación de visualización. 2. Termina el escenario.	

Flujos alternos		
Relaciones	CU incluidos	No Aplicable
	CU extendidos	No Aplicable
Requisitos no funcionales	No Aplicable	
Asuntos pendientes	No Aplicable	

Caso de Uso No. 09. Aplicar Actualización de Módulos.	
Objetivo	Aplicar nuevas actualizaciones para los componentes instalados.
Actores	Reloj
Resumen	El sistema aplica las actualizaciones para los componentes instalados.
Complejidad	Baja
Prioridad	Auxiliar
Referencias	RF 4.1. RF 4.2
Precondiciones	El sistema debe haber encontrado actualizaciones
Postcondiciones	
Flujo de eventos	
Flujo básico	
1. El sistema busca nuevas actualizaciones para los componentes instalados en la estación de visualización. 2. El sistema aplica las actualizaciones encontradas a los componentes de la estación de	

visualización. 3. Termina el escenario.		
Flujos alternos		
Relaciones	CU incluidos	CU Buscar Actualización de Módulos.
	CU extendidos	No Aplicable
Requisitos no funcionales	No Aplicable	
Asuntos pendientes	No Aplicable	

Caso de Uso No. 10. Aplicar Configuración de Módulos.	
Objetivo	Aplicar nuevas configuraciones para los componentes instalados.
Actores	Herramienta de Configuración.
Resumen	El sistema aplica las configuraciones definidas en la Herramienta de Configuración para los componentes instalados.
Complejidad	Baja
Prioridad	Auxiliar
Referencias	RF 5.1
Precondiciones	
Postcondiciones	
Flujo de eventos	
Flujo básico	
1. La herramienta de configuración envía la nueva configuración para los componentes al	

servicio. 2. El sistema aplica la configuración a los componentes instalados en la estación de visualización. 3. Termina el escenario.		
Flujos alternos		
Relaciones	CU incluidos	CU Buscar Actualización de Módulos.
	CU extendidos	No Aplicable
Requisitos no funcionales	No Aplicable	
Asuntos pendientes	No Aplicable	

En este capítulo se propuso la implementación de una aplicación que logre integrar de forma desacoplada, comunicar y configurar los diferentes módulos del alas PACS, en una estación de visualización. Para un mayor entendimiento del flujo de eventos, se definió el modelo de dominio del alas PACSManager Además se definieron los requisitos funcionales y no funcionales del alas PACSManager así como los casos de uso del sistema con sus descripciones expandidas.

CAPÍTULO 3. Análisis y Diseño del Componente

En este capítulo se definen las clases de diseño, sus atributos y responsabilidades. Se analizará la arquitectura que tiene el sistema para el cumplimiento de sus requerimientos y se expondrán los distintos diagramas de clases de diseño y secuencia con el objetivo de un mejor entendimiento del sistema desde su núcleo.

3.1 Modelo Arquitectónico.

El sistema está basado en mensajes el cual se comporta como el estilo arquitectónico de bus de mensajería (BSE), este consiste en un combinado de arquitectura de software que proporciona servicios fundamentales para arquitecturas complejas a través de un sistema de mensajes (el bus) basado en las normas y que responde a eventos.

Un Bus de servicio de empresa (BSE) es una infraestructura de software que funciona como capa intermedia (middleware), proporcionando servicios de integración de las distintas aplicaciones a través de mensajería basada en estándares y servicios de sincronización. Aunque un BSE no implementa por sí mismo una arquitectura orientada a servicios (SOA), proporciona características para su implementación.

BSE proporciona una capa de abstracción construida sobre una implementación de un sistema de mensajes de empresa que permita a los expertos en integración explotar el valor del envío de mensajes sin tener que escribir código. Sus principales ventajas son: acomodación de sistemas existentes más rápida y barata, mayor flexibilidad (más fácil de cambiar si hay nuevos requisitos), basado en normas o reglas, posibilidad de escalar desde soluciones puntuales hasta implementaciones de empresa (bus distribuido), tipos de servicio listos-para-funcionar (*ready-to-use*) predefinidos, mayor configuración en vez de tener que codificar la integración, entre otras.

Este estilo soporta el uso de patrones como: Inversión por Control (IoC) el cual es un patrón de diseño pensado para permitir un menor acoplamiento entre componentes de una aplicación y fomentar así la reutilización de los mismos, además del patrón Inyección de Dependencia el cual es cuando una clase principal hace uso de una clase secundaria sin hacer referencia directa a ella. Alguna entidad externa proveerá la clase secundaria a la clase principal en tiempo de ejecución, inyectará la dependencia. [17]

3.2 Diseño.

En la fase de diseño se modela el sistema y se encuentra su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Una entrada esencial en el diseño es el resultado del análisis. En la fase de diseño se logran objetivos puntuales y de vital importancia para el futuro resultado de la investigación tales como: transformar los requerimientos a diseños del sistema, desarrollar una arquitectura robusta para el sistema, adaptar el diseño para hacerlo corresponder con el ambiente de implementación. [18]

El modelo del diseño describe la realización de los casos de uso, donde intervienen los diagramas de clases de diseño y los diagramas de colaboración y/o secuencia.

Los diagramas de clases son usados para mostrar las clases y paquetes que conforman el sistema. Estos diagramas brindan una vista estática de los elementos lógicos que componen un sistema y sus relaciones. [19]

Para un mejor entendimiento de los requerimientos y de los procesos planteados para el desarrollo de esta investigación expresados en clases de diseño se realizaron los diagramas de clases del diseño. Ver Anexo (Diagramas de Clases)

Un diagrama de secuencia muestra las interacciones entre objetos ordenadas en secuencia temporal; muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario. [20]

Ver Anexo(Diagramas de Secuencias)

3.2.1 Descripción de las clases del diseño

3.2.1.1 Clase Controller.

Nombre: Controller	
Tipo de clase: controladora	
Atributo	**Tipo**
_systemSection	SystemSection
ExitCommand	ICommand
StartCommand	ICommand

Para cada responsabilidad:	
Nombre:	Controller()
Descripción:	Es el constructor de la clase.
Nombre:	CreateShell()
Descripción:	
Nombre:	BuildMenuItem()
Descripción:	
Nombre:	ConfigureCatalog(AgregateCatalog)
Descripción:	
Nombre:	CreateShellMenu()
Descripción:	
Nombre:	InitializeShell()
Descripción:	
Nombre:	OnDispose()
Descripción:	
Nombre:	Startup()
Descripción:	

3.2.1.2 Clase Metadatos

Nombre: BalloonTip	
Tipo de clase: entidad	
Atributo	**Tipo**
_taskBarIcon	TaskBarIcon
Name	string
Para cada responsabilidad:	
Nombre:	BalloonTip()
Descripción:	Es el Constructor de la clase.
Nombre:	ShowBalloonTip(string, string,BalloonIcon)
Descripción:	Muestra el icono del servicio.

Nombre:	ContexMenu()
Descripción:	Llena el menú asociado al ícono del servicio.
Nombre:	MouseDoubleClick(object, RoutedEventArgs)
Descripción:	
Nombre:	ShowCustomBalloon (UIElement, PopupAnimation, int)
Descripción:	

3.2.1.3 Clase SystemElementCollection.

Nombre: SystemElementCollection	
Tipo de clase: entidad	
Atributo	**Tipo**
ElemName	string
Para cada responsabilidad:	
Nombre:	Add(SystemElement)
Descripción:	Adiciona un elemento de configuración a una lista.
Nombre:	CreateNewElement()
Descripción:	Crea un elemento de configuración.
Nombre:	GetElementKey(ConfigurationElement)
Descripción:	Devuelve un elemento de configuración.
Nombre:	Remove(SystemElement)
Descripción:	Elimina un elemento del sistema.

3.2.1.4 Clase App.

Nombre: App	
Tipo de clase: entidad	
Atributo	**Tipo**
_controller	Controller
Para cada responsabilidad:	
Nombre:	OnExit(ExitEventArg)
Descripción:	Cierra el servicio.

Nombre:	OnStartup(StartupEventArg)
Descripción:	Inicia el servicio.

3.2.1.5 Clase SystemManager.

Nombre: SystemManager	
Tipo de clase: entidad	
Atributo	**Tipo**
_current	ISystemManager
_systems	NotificationCollection<ISystem>
Para cada responsabilidad:	
Nombre:	SystemManager()
Descripción:	Es el constructor de la clase.
Nombre:	CreateInstance ()
Descripción:	Crea una instancia.
Nombre:	GetSystems ()
Descripción:	Obtiene una instancia.
Nombre:	RegisterSystem()
Descripción:	Registra una instancia cargada.
Nombre:	UnRegisterSystem()
Descripción:	Elimina una instancia de la lista de instancias registradas.
Nombre:	SystemSendMessage(SystemMessage)
Descripción:	Envía un mensaje a una instancia.
Nombre:	StartSystem(string)
Descripción:	Inicializa una instancia.

3.2.1.6 ISystemManager

Nombre: ISystemManager	
Tipo de clase: Interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	

Nombre:	StartSystem (string)
Descripción:	Inicializa una instancia.

3.2.1.7 Clase IMessageSender

Nombre: IMessageSender	
Tipo de clase: interfaz	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	SendMessage (SystemMessage)
Descripción:	Envía un mensaje.

En este capítulo se identificó el estilo arquitectónico Bus de Mensajería como arquitectura de análisis y diseño del alas PACSManager. Además se mostró una vista de las diferentes clases del diseño asi como su descripción para su posterior implementación.

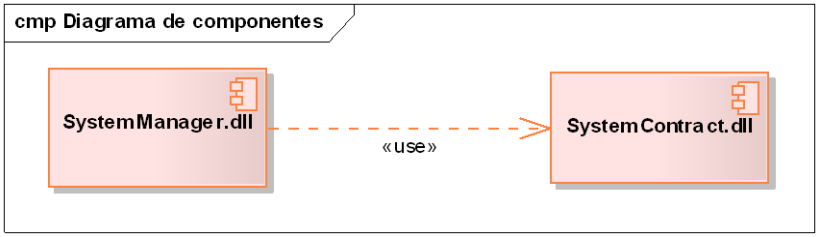
CAPÍTULO 4. Implementación del Componente

En el presente capítulo se especificarán los componentes ejecutables del sistema, así como su interacción, quedando conformado de esta forma el diagrama de componentes correspondiente al flujo de trabajo implementación.

4.1 Diagrama de Componentes.

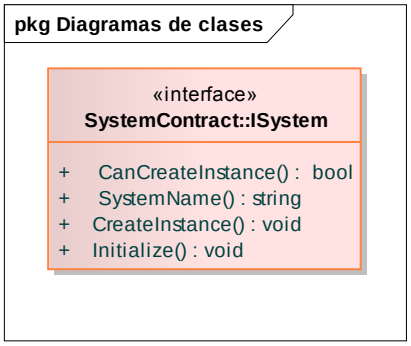
Un componente representa una parte física del sistema, los mismos pueden ser una librería, un ejecutable, una tabla etc. Estos engloban la implementación de un grupo de clases del diseño, además de que cada uno de ellos define una interface que describe su funcionalidad y forma de empleo.

El diagrama de componentes permite conocer a los desarrolladores la estructura física que tiene el sistema y como se relacionan sus partes.



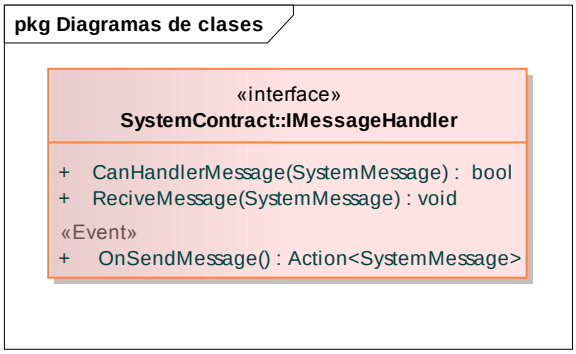
4.2 Caso de Estudio.

Para la incorporación de un nuevo componente al alas PACS se deben implementar las interfaces ISystem, IMessageSender y IMessageHandler que se encuentran en las librerías (dll) de SystemContract, además de hacer referencia a las librerías DAF y HardCode.Wpf.TaskBarNotification existentes en la solución alas PACS.



ISystem: esta interfaz posee los métodos que permiten el trabajo con una instancia del nuevo componente.

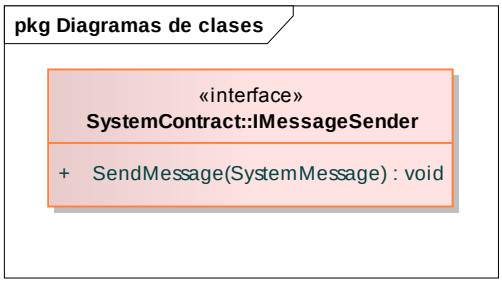
- ✓ CreateInstance () para la creación de un nuevo componente.
- ✓ Initialize () que permite la inicialización del nuevo componente
- ✓ SystemName () que devuelve el nombre del nuevo componente.
- ✓ CanCreateInstance () que permite saber si se puede crear una nueva instancia del componente nuevo.



IMessageHandler: esta interfaz posee los métodos que permiten ejecutar acciones sobre el nuevo componente.

- ✓ CanHandlerMessage () que recibe por parámetro un mensaje de un módulo, permite saber si se puede ejecutar la acción enviada desde otro módulo hacia este nuevo componente.
- ✓ ReciveMessage () que recibe por parámetro un mensaje de un módulo, para el recibo de mensajes desde otros módulos hacia el nuevo componente.

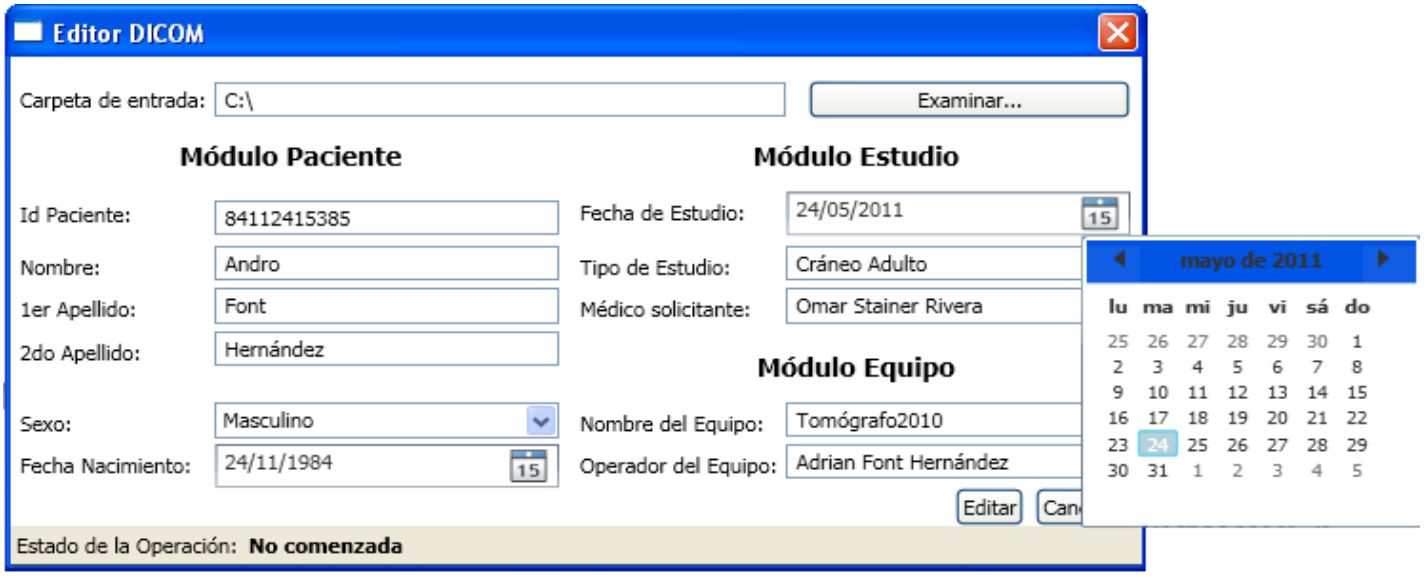
- ✓ OnSendMessage este es un evento que constituye la acción tomada por el nuevo componente para dar respuesta a las peticiones recibidas desde otra instancia de módulo.



IMessageSender: esta interfaz permite el envío de mensajes hacia otras instancias de módulos a través del método:

- ✓ SendMessage ().

El siguiente ejemplo muestra la incorporación de un nuevo componente a la solución alas PACS. Este nuevo componente constituye un Editor de DICOM (Editor DICOM) el cual fue constituido para hacer pruebas reales al sistemas alas PACS y su principal Requisito funcional es editar los metadatos de una imagen DICOM.



Ejemplo de Implementación:

Para la incorporación de este nuevo componente a la solución alas PACS se debe implementar las interfaces antes mencionadas:

```
[Export(typeof(ISystem))]


public partial class WinMain : Window, IMessageHandler, ISystem
```

La expresión `[Export(typeof(ISystem))]` permite establecer un contrato con la solución alas PACS como componente del sistema y así poder interactuar con los demás componentes del mismo.

Implementación de la interfaz ISystem

```
# region ISystem Members

        public bool CanCreateInstance
        {
            get { return true; }
        }

        public string SystemName
        {
            get { return "DicomEditor"; }
        }

        public void Initialize()
        {
            _messageHandler.RegisterMessage("OpenDicomEditor", (Action<DirectoryInfo>)
OpenDicomEditor);


        public void CreateInstance()
        {
            Show();
        }

        public void Dispose()
        {

        }
```

```
        # endregion
```

Implementación de la interfaz IMessageHandler

```
# region IMessageHandler

        public bool CanHandlerMessage(SystemMessage message)
        {
            return _messageHandler.CanHandlerMessage(message);
        }

        public void ReceiveMessage(SystemMessage message)
        {
            _messageHandler.ReceiveMessage(message);
            tbInputFolder.Text = message.Params[0].ToString();
        }

        public event Action<SystemMessage> OnSendMessage;


        # endregion
```

4.3 Pruebas de Rendimiento

Las pruebas realizadas a la aplicación alas PACSManager se hicieron incorporando dll del Sistema Operativo Windows XP al directorio raíz de la solución alas PACS con un promedio de peso de las dll de 0.36 Megabyte (Mb), actualmente la solución alas PACS cuenta con 92 archivos de ellas solo 47 son dll entre módulos y librerías, las cuales tiene un promedio de peso de archivos de 0.34 Mb.

Para las pruebas de rendimiento se utilizó una computadora con 1 Giga Byte de memoria RAM, en un procesador Pentium a 3 Giga Hertz de velocidad.

Cantidad de dll	Tiempo(ms)
10	15
20	17
140	28
140	5

Tabla 4.1 Tabla de Rendimiento.

Las pruebas de rendimiento realizadas al componente alas PACSManager arrojaron como resultado que este puede cargar una gran cantidad de módulos en muy poco tiempo (tabla 4.1). La primera vez en ejecutarse este reconoce cuales de los módulos en su directorio raíz pertenecen a la solución alas PACS y cuáles no, en las próximas ejecuciones del alas PACSManager, este, ya conoce cuales forman parte del alas PACS y la carga de módulos se realiza de forma más rápida.

CONCLUSIONES

Una vez concluida la investigación sobre sistemas para la integración, manejo y configuración de aplicaciones. Se le ha dado cumplimiento a los objetivos planteados y se obtuvieron los resultados que a continuación se mencionan:

Las soluciones existentes para la orquestación de acciones en un sistema informático, no responden a la necesidad de integración desacoplada, manejo y configuración de los módulos del alas PACS en una estación de visualización.

Después de un análisis de las tecnologías para la implementación de soluciones de este tipo se seleccionaron las tecnologías MEF y WCF para la implementación del componente alas PACSManager. Por las características del alas PACSManager, para su desarrollo se seleccionó Bus de Mensajería como arquitectura de diseño e implementación.

Para el desarrollo del componente y para futuras versiones, se generaron las plantillas correspondientes a los flujos de trabajo Requerimientos, Análisis y Diseño e Implementación.

Como resultado final, se obtuvo un servicio capaz de manejar, configurar e integrar, de forma desacoplada, los diferentes módulos del alas PACS en una estación de visualización.

RECOMENDACIONES

Para darle continuidad a este trabajo en aras de mejorar sus prestaciones y contribuir con los servicios que brinda el alas PACS, los autores recomiendan:

Instalar el componente alas PACSManager en instituciones médicas donde se encuentre desplegada la solución alas PACS.

Mantener un proceso continuo de atención a los usuarios del sistema alas PACS para obtener una retroalimentación que facilite la captación de nuevos requerimientos y funcionalidades para elevar la calidad del componente alas PACSManager.

Revisar sistemáticamente el estado del arte de los estándares y tecnologías vinculadas a la comunicación e integración de subsistemas para mantener actualizado el componente alas PACSManager.

REFERENCIAS BIBLIOGRÁFICAS

[1] , [En línea] http://cnx.org/content/m17405/latest/.

[2] Moliner, Maria., *Diccionario de María Moliner.*

[3] Tedeschi, Nicolás., [En línea] http://msdn.microsoft.com/es-es/library/bb972240.aspx)..

[4] , *(Marileydis Molina Alonso Metodología para el relevamiento de información en Instituciones Hospitalarias para el despliegue de sistemas PACS - RIS.*

[5] [En línea] http://msdn.microsoft.com/es-es/library/aa559390%28v=BTS.10%29.aspx.

[6] [En línea] http://geeks.ms/blogs/ciin/archive/2009/04/23/managed-extensibility-framework-un-ejemplo-pr-225-ctico-i.aspx.

[7] [En línea] http://www.matthidinger.com/archive/2008/10/12/managed-addin-framework-system.addin-with-wpf.aspx.

[8] [En línea] http://social.msdn.microsoft.com/Search/es-es?query=Remoting.

[9] [En línea] http://www.freedesktop.org/Software/dbus.

[10] [En línea] http://msdn.microsoft.com/es-es/library/ms735119.aspx.

[11] *Aplicación de la Metodología RUP para el Desarrollo Rápido de Aplicaciones Basado en el Estándar J2EE.*

[12] [En línea] jovenclub.cuhttp://gutl.jovenclub.cu/aplicaciones.

[13] [En línea] http://www.sparxsystems.com.ar/products/ea.html.

[14] [En línea] http://msdn.microsoft.com/en-us/magazine/ee336128.aspx.

[15] [En línea] http://adimen.si.ehu.es/~rigau/teaching/EHU/ISHAS/Curs2008-2009/Apunts/IS.4.pdf.

[16] , "ISW_UCI." 2008-2009.

[17] , [En línea] (http://www.buenastareas.com/ensayos/Enterprise-Service-Bus/1793587.html(bus de mensaje).

[18] scribd.com. [En línea] http://www.scribd.com/doc/7978336/Ingenieria-de-Software-Un-Enfoque-Practico-Pressman-5th-Ed.

[19] slideshare.net. [En línea] http://www.slideshare.net/coordinacionylaboratorios/fase-de-diseo-y-analisis-de-datos.

[20] inti.gob.ar. [En línea] http://www.inti.gob.ar/prodiseno/pdf/n141_proceso.pdf.

BIBLIOGRAFÍA

Aplicación de la Metodología RUP para el Desarrollo Rápido de Aplicaciones Basado en el Estándar J2EE.

http://adimen.si.ehu.es/~rigau/teaching/EHU/ISHAS/Curs2008-2009/Apunts/IS.4.pdf.

http://geeks.ms/blogs/ciin/archive/2009/04/23/managed-extensibility-framework-un-ejemplo-pr-225-ctico-i.aspx.

http://msdn.microsoft.com/en-us/magazine/ee336128.aspx.

http://social.msdn.microsoft.com/Search/es-es?query=Remoting.

http://www.freedesktop.org/Software/dbus.

http://www.matthidinger.com/archive/2008/10/12/managed-addin-framework-system.addin-with-wpf.aspx.

http://www.scribd.com/doc/7978336/Ingenieria-de-Software-Un-Enfoque-Practico-Pressman-5th-Ed.

ISW_UCI. 2008-2009.

Marileydis Molina Alonso, Metodología para el relevamiento de información en Instituciones Hospitalarias para el despliegue de sistemas PACS - RIS.

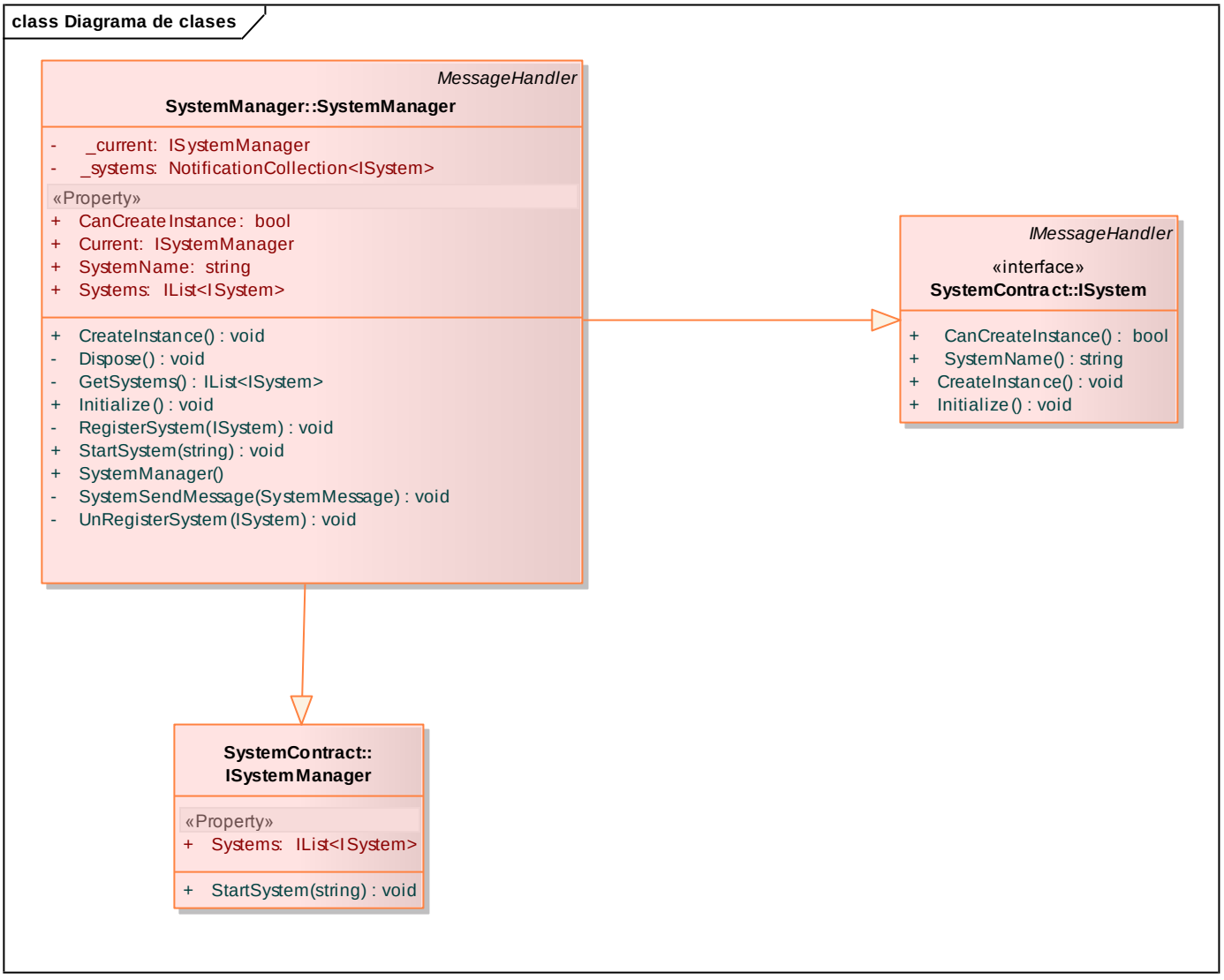
Moliner, Maria., Diccionario de María Moliner.

Tedeschi, Nicolás., [En línea] http://msdn.microsoft.com/es-es/library/bb972240.aspx)..

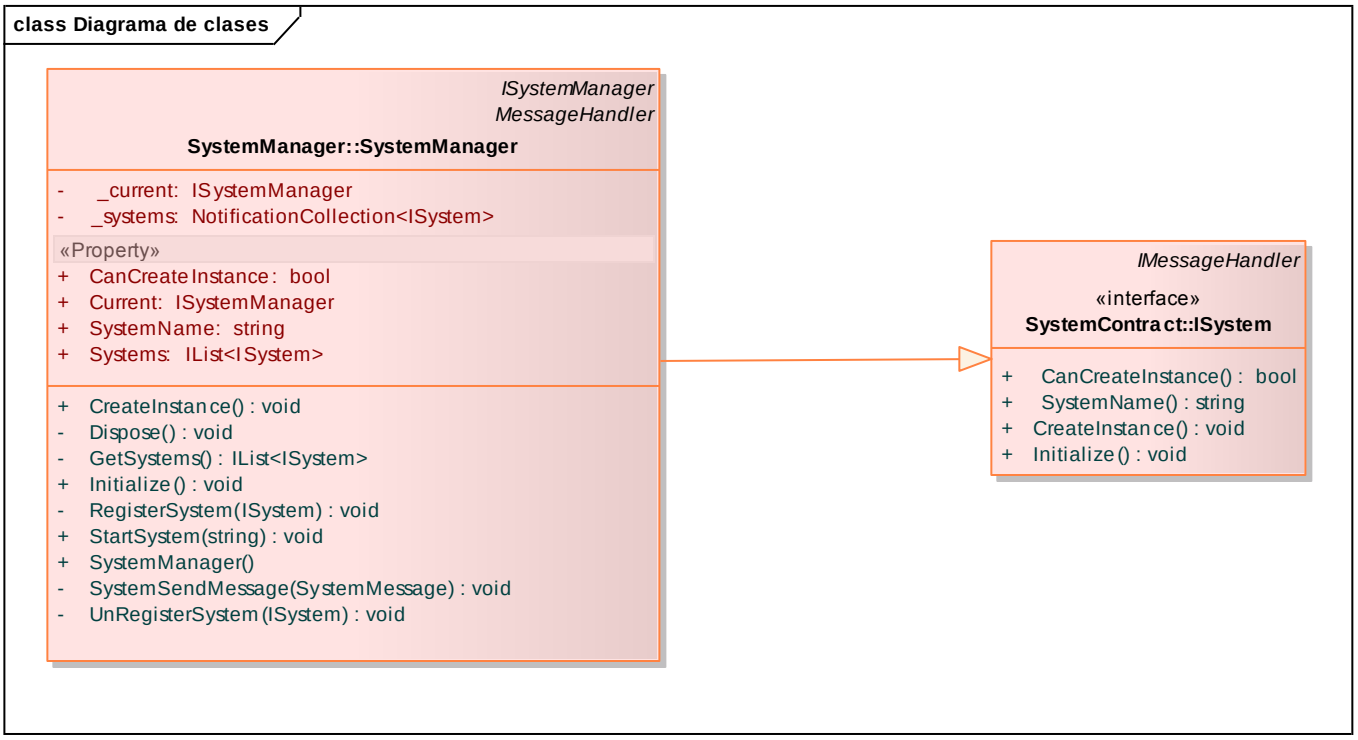
ANEXOS

Diagrama de Clases del Diseño

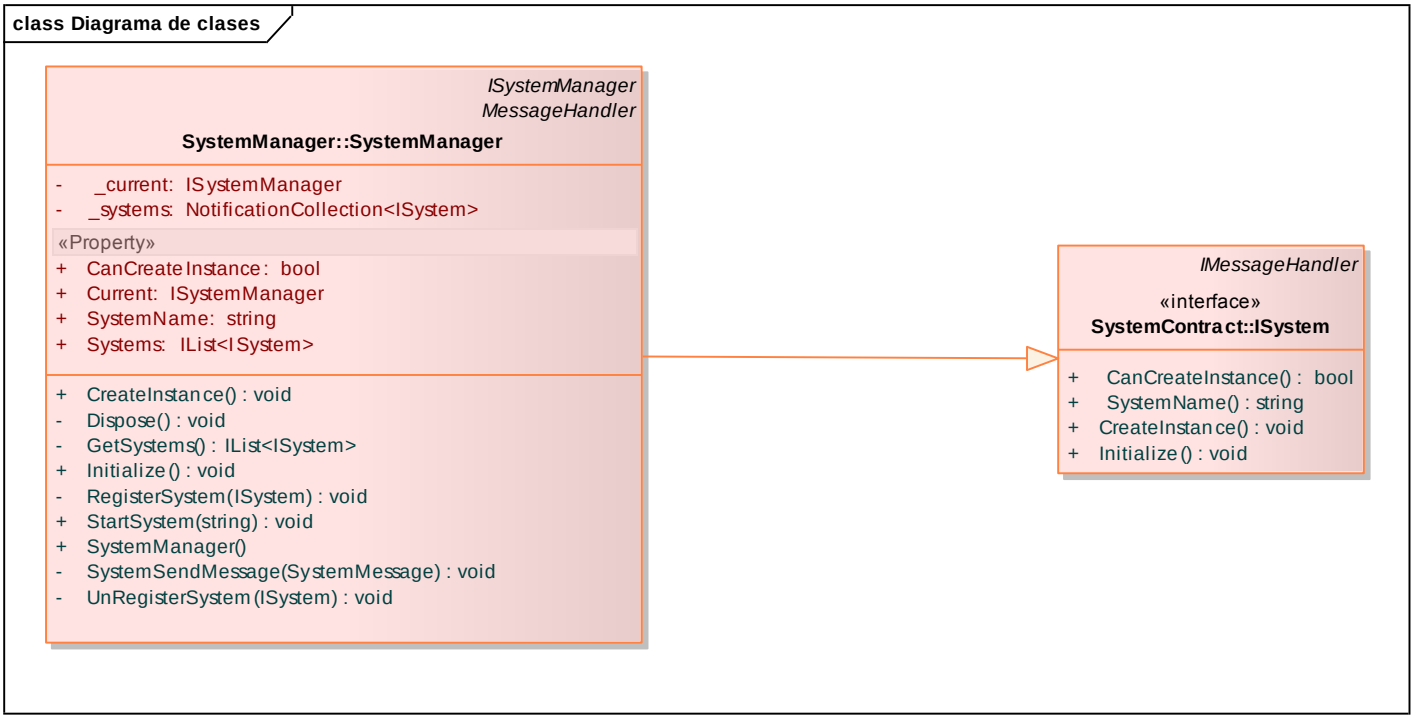
Caso de Uso Cargar Módulo.



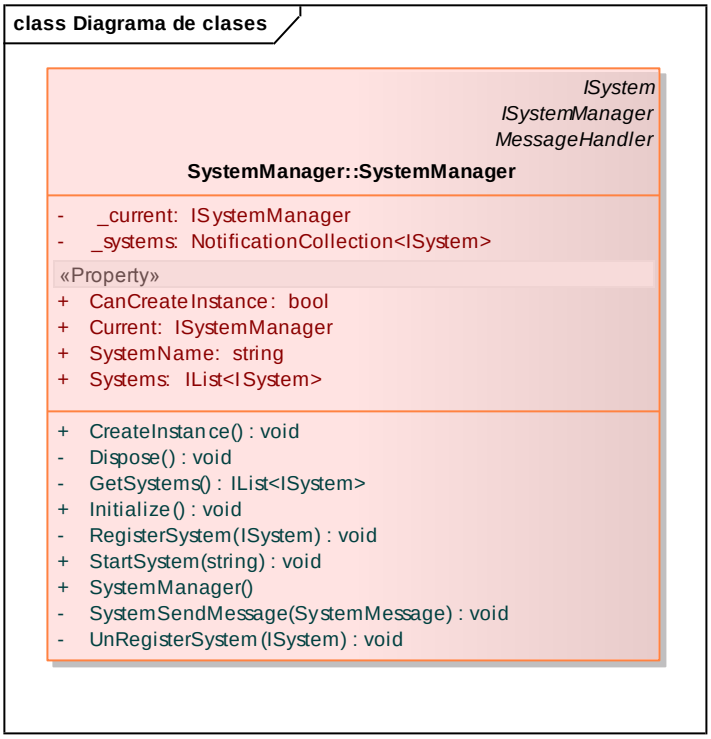
Caso de Uso Validar Certificado.



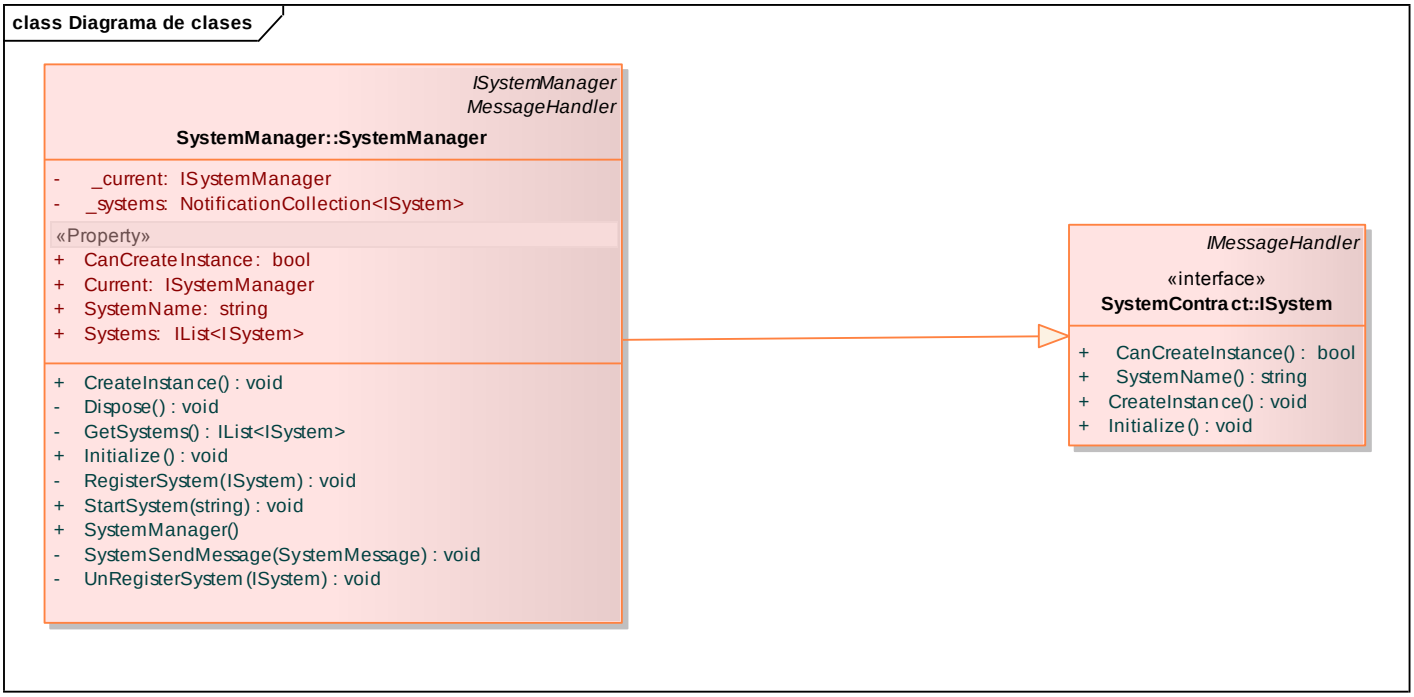
Caso de Uso Enviar Mensaje.



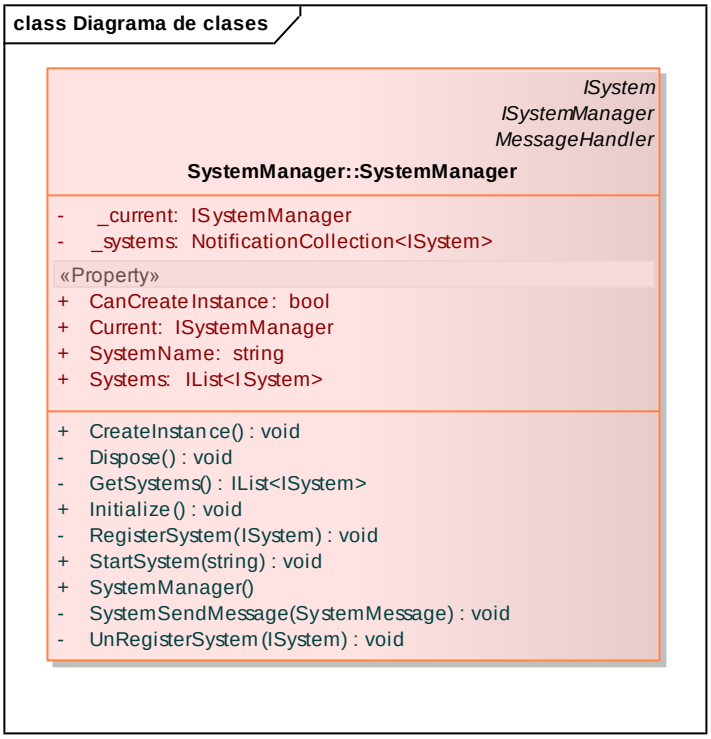
Caso de Uso Recibir Mensaje.



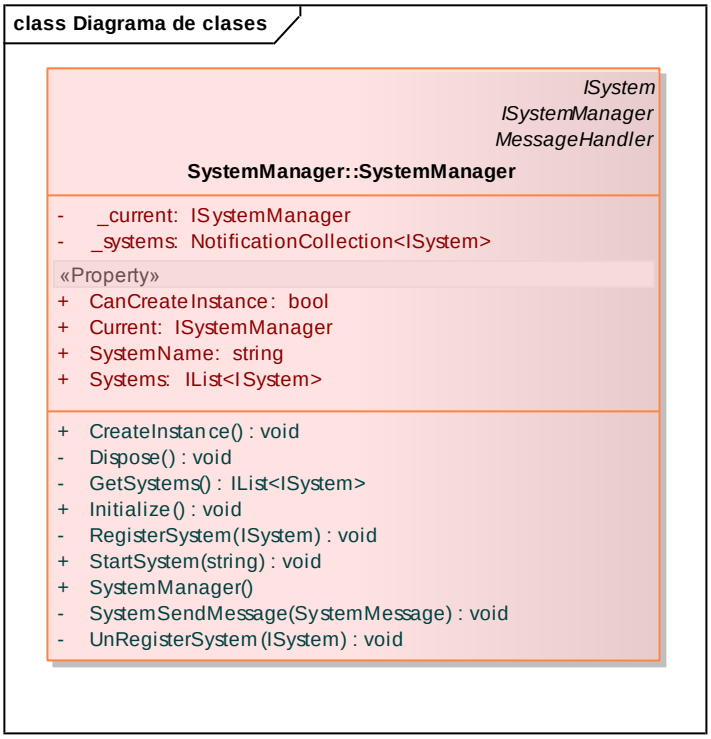
Caso de Uso Registrar Instancia.



Caso de Uso Cerrar Instancia.

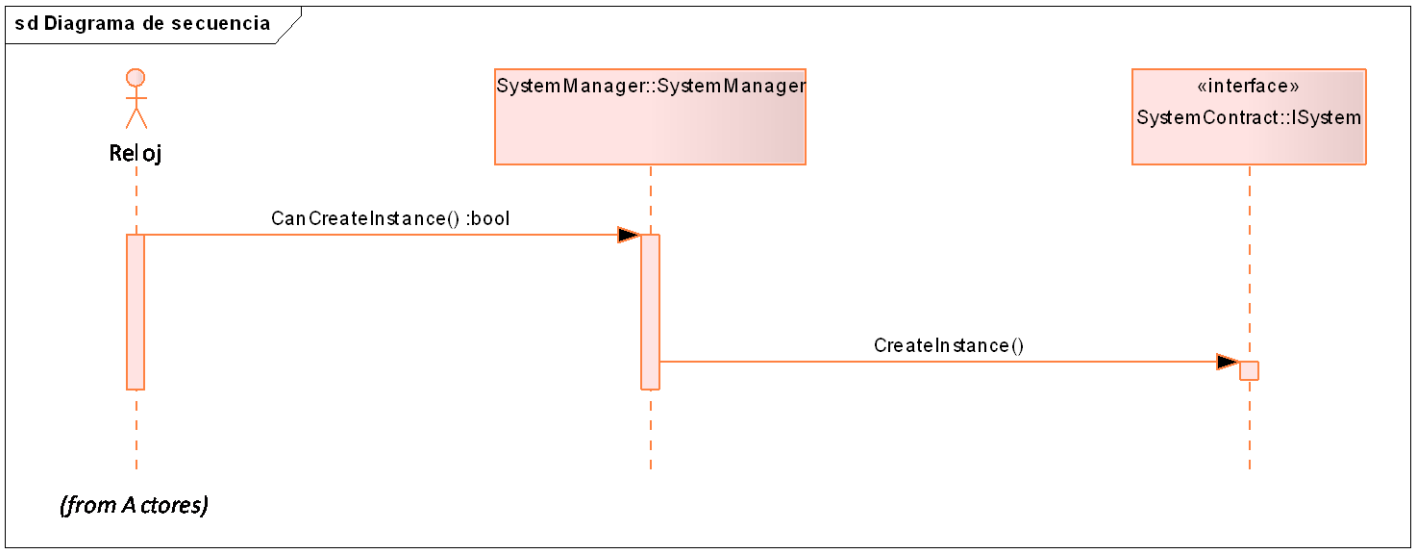


Caso de Uso Buscar Instancia.

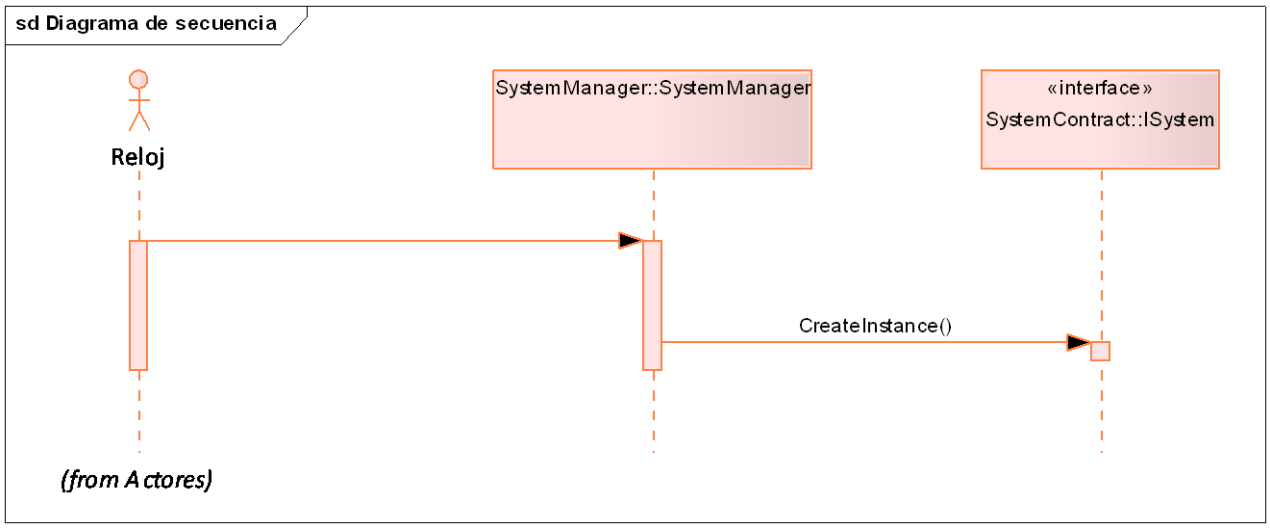


Diagramas de Secuencia.

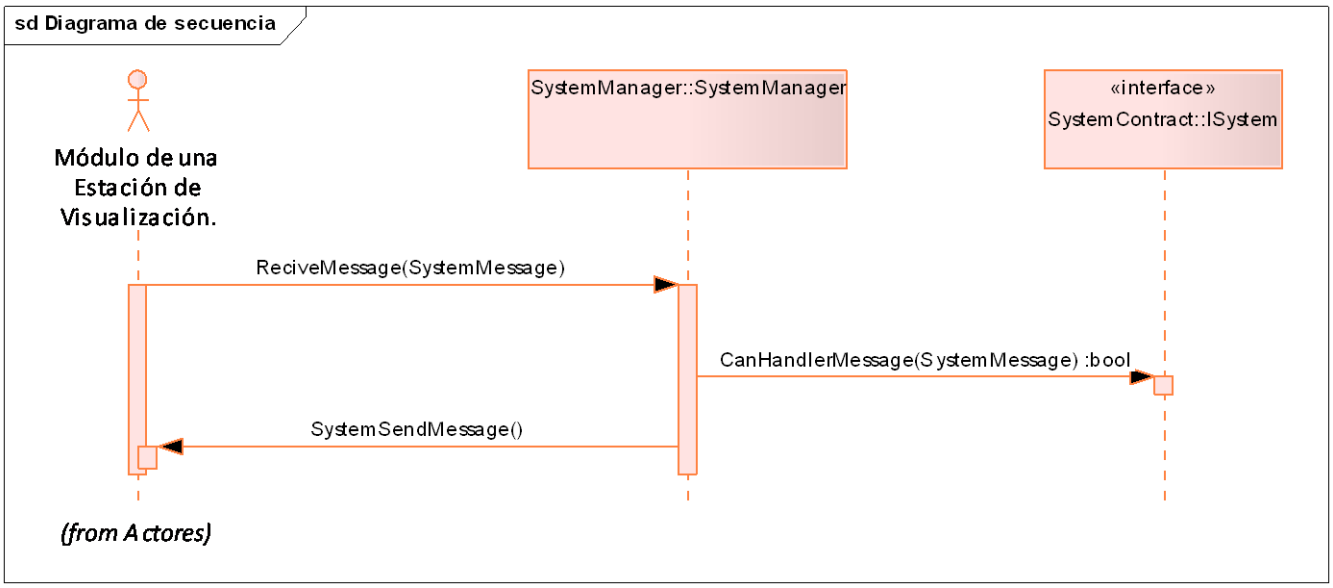
Caso de Uso Cargar Módulo.



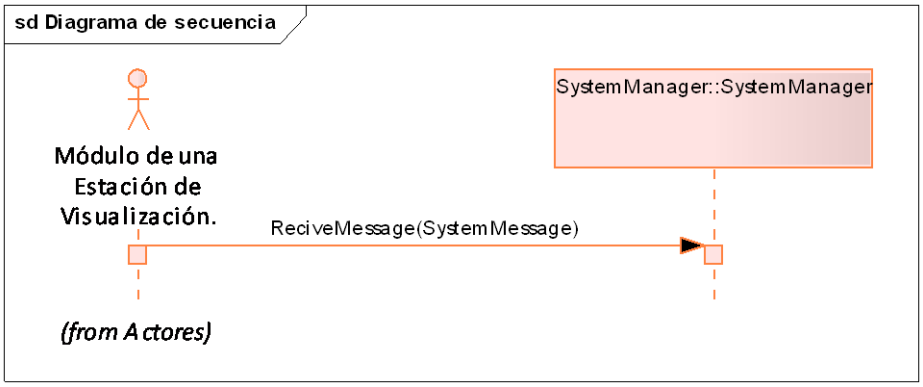
Caso de Uso Validar Certificado.



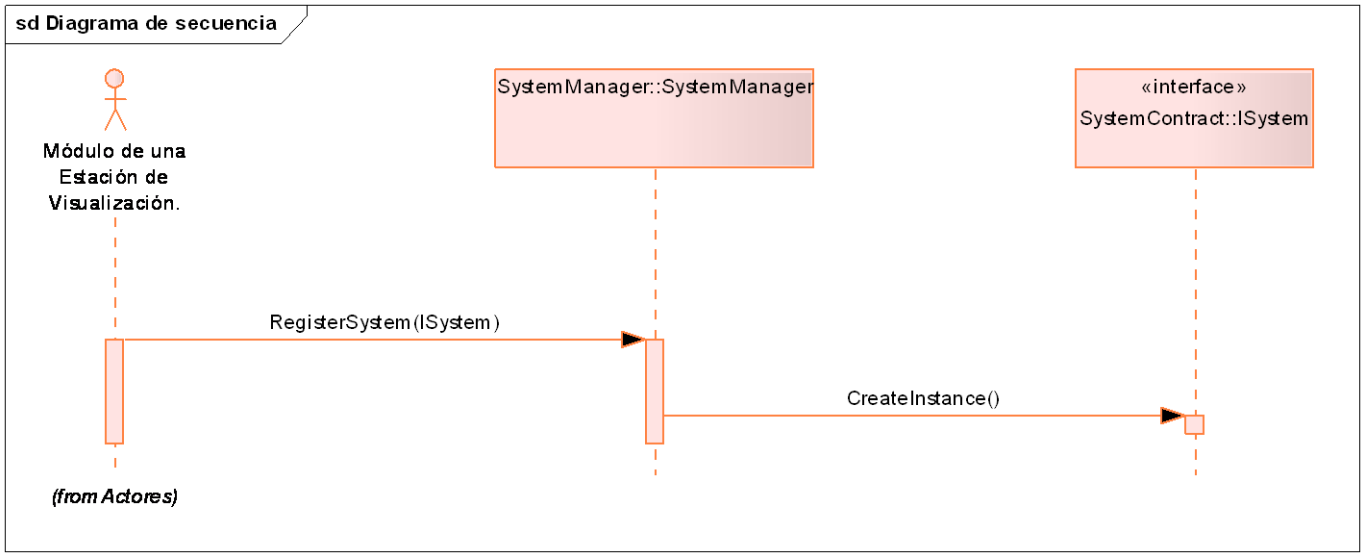
Caso de Uso Enviar Mensaje.



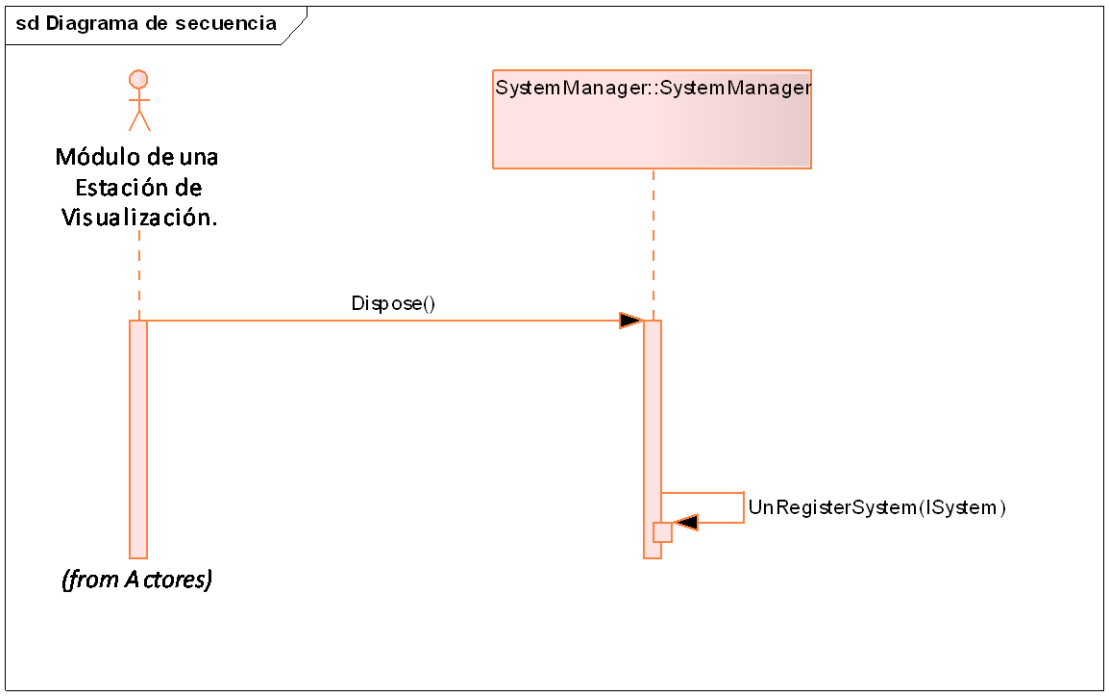
Caso de Uso Recibir Mensaje.



Caso de Uso Registrar Instancia.

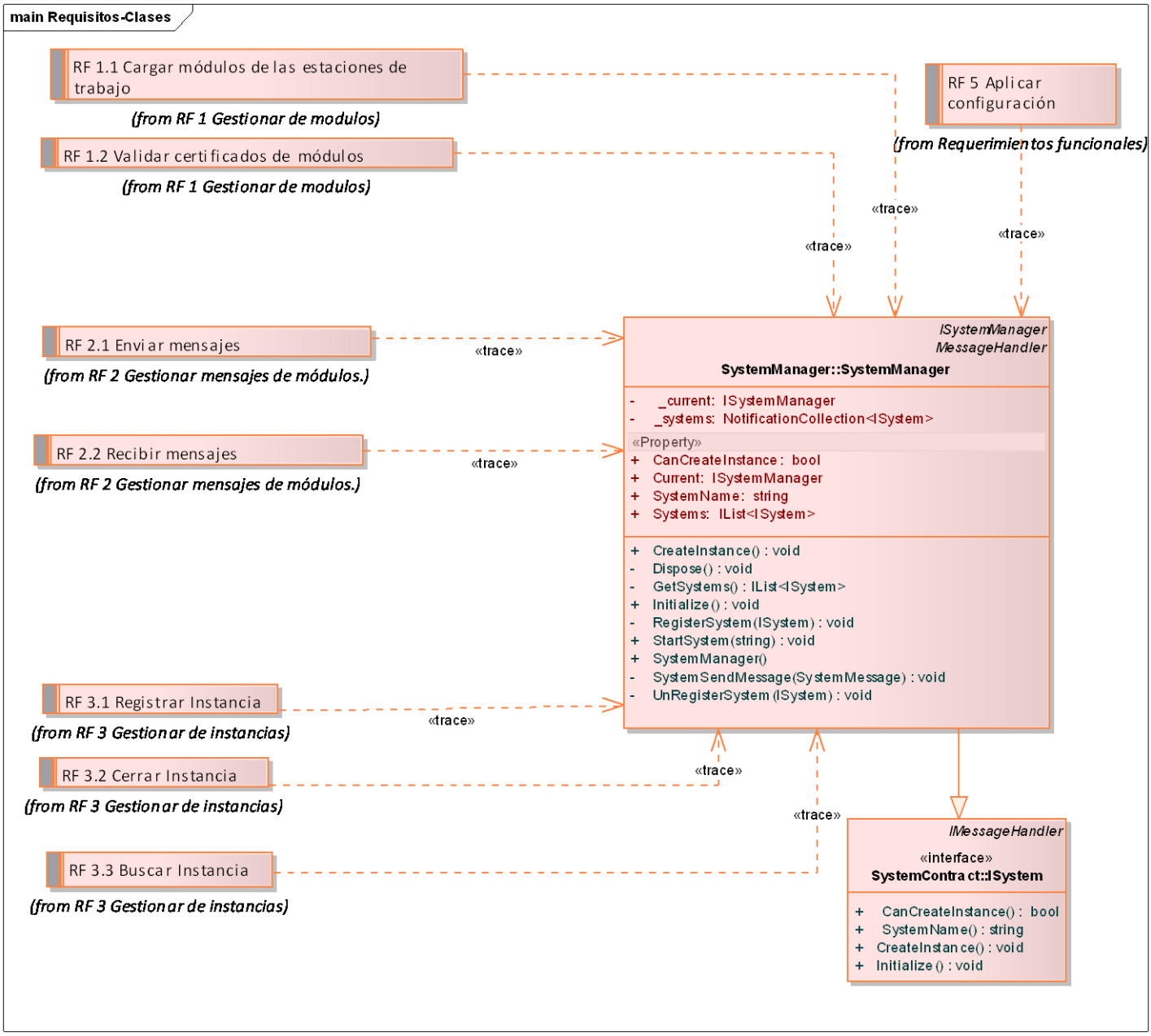


Caso de Uso Cerrar Instancia.

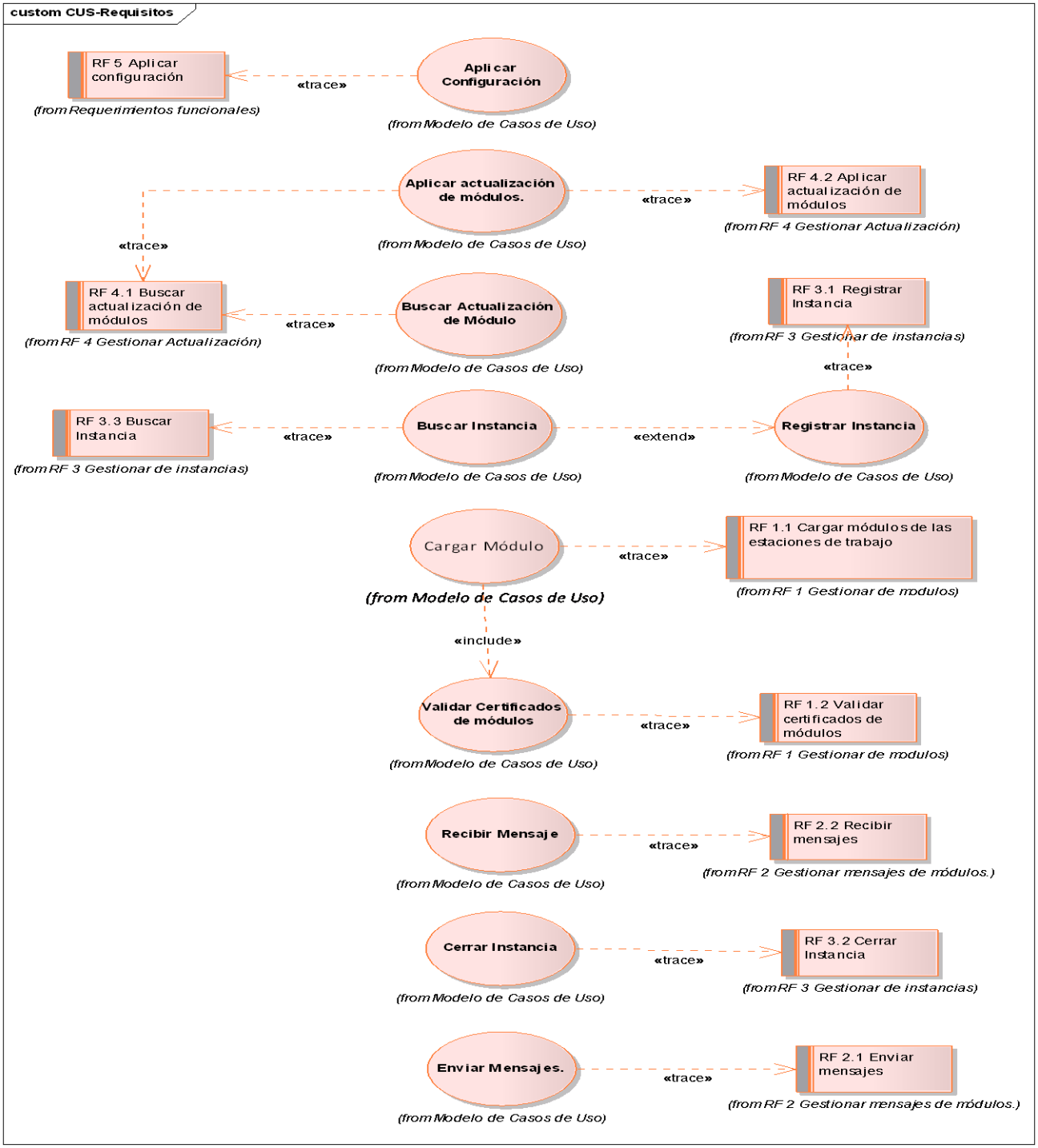


Diagramas de Trazabilidad.

Requisitos – Clases.



Casos de Uso – Requisitos.



GLOSARIO DE TÉRMINOS

.NET: es una plataforma de Microsoft que provee un extenso conjunto de soluciones predefinidas para necesidades generales de la programación de aplicaciones, y administra la ejecución de los programas escritos específicamente con la plataforma.

API: Application Programming Interface es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usados generalmente en las bibliotecas.

Archivo .msi: Es un archivo instalador de Windows, usado para instalar aplicaciones. El nombre de la extensión msi proviene de Microsoft Installer (Instalador de Microsoft). Al hacer doble click sobre un archivo msi éste se abre (ejecuta) con Windows Installer y no tiene un editor asociado.

DICOM: Digital Imaging and Communications in Medicine. Estándar para el tratamiento de imágenes digitales y comunicaciones para el campo de la medicina, que facilita el manejo de la información médica entre hospitales y centros de investigación.

DLL: Dynamic Link Library (biblioteca de enlace dinámico) son archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo. Esta denominación es exclusiva a los sistemas operativos Windows siendo ".dll" la extensión con la que se identifican estos ficheros, aunque el concepto existe en prácticamente todos los sistemas operativos modernos.

Ensamblado: Es una colección de tipos y recursos creados para funcionar en conjunto y formar una unidad lógica de funcionalidad. Los ensamblados proporcionan la información necesaria para conocer las implementaciones de tipos. Para el motor en tiempo de ejecución, un tipo no existe si no es en el contexto de un ensamblado. Constituye la unidad fundamental de implementación, control de versiones, reutilización, ámbitos de activación y permisos de seguridad.

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. Incluye soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

HIS: Sistema de Información Hospitalaria. Son sistemas de software que trabajan acoplados y permiten la informatización de los distintos servicios de las instituciones de salud.

HTTP: HyperText Transfer Protocol (Protocolo de transferencia de hipertexto). Es el protocolo usado para intercambiar archivos (texto, gráfica, imágenes, sonido, video y otros archivos multimedia) en la World Wide Web.

Metadatos: Son «descripciones estructuradas y opcionales que están disponibles de forma pública para ayudar a localizar objetos» o «datos estructurados y codificadas que describen características de instancias conteniendo informaciones para ayudar a identificar, descubrir, valorar y administrar las instancias descritas».

PACS: Picture Archiving and Communication System (PACS). Sistema para el almacenamiento y comunicación de imágenes médicas.

Plugins: 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 interactúan por medio de la API.

RIS: Sistema de Información Radiológica (Radiological Information System). Es un sistema encargado de la gestión de la información generada y manipulada como resultado de los procesos de negocio de carácter radiológico (imagenológico).

TCP: Transmission Control Protocol (en español *Protocolo de Control de Transmisión*). Es un protocolo de comunicación orientado a conexión y fiable del nivel de transporte.

WSDL: Web Services Description Language, un formato XML que se utiliza para describir servicios Web (algunas personas lo leen como wisdel).

XML: Extensible Markup Language es un lenguaje de marcado extensible que puede usarse para almacenar datos en un formato estructurado, basado en texto y definido por el usuario.