Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA- IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

DISEÑO DE UN NÚCLEO DE ARQUITECTURA COMÚN PARA CONSTRUIR PRODUCTOS DE PACS.

Tesis que presenta:
Marco Antonio Núñez Gaona.
Para obtener el grado de:
Maestro en Ciencias y Tecnologías de la Información.

Asesores:

Dr. Humberto Cervantes Maceda.

M. en C. Alfonso Martínez Martínez.

Jurado Calificador:

Presidente: Dr. Juan Ramón Jiménez Alaníz.

Secretario: Dra. Perla Inés Velasco Elizondo.

Vocal: Dr. Humberto Cervantes Maceda.

México, D. F. Julio 2010.

“Every solution breeds new problems.”
Murphy´s Law.

A mis padres y familia
que siempre confiaron en mi.

AGRADECIMIENTOS

A mis asesores de tesis, Dr. Humberto Cervantes Maceda y M.C. Alfonso Martínez Martínez por su gran apoyo y paciencia en el desarrollo de este trabajo.

A la Dra. Josefina Gutiérrez Martínez, por el apoyo recibido en el Instituto Nacional de Rehabilitación.

A mi esposa María Antonia Pérez Jiménez y a mi hijo Ramsés Núñez Pérez por su comprensión y gran apoyo.

A mis amigos, por sus consejos.

Contenido

1 Introducción.

En la actualidad, los sistemas de software son el pilar fundamental sobre el cual se sostienen gran cantidad de organizaciones modernas (hospitales, aeropuertos, casas de bolsa, etc.). Estos sistemas permiten administrar información que es utilizada como materia prima en las organizaciones. Muchos de los sistemas que se desarrollan actualmente son altamente complejos; el desarrollo de estos sistemas requiere de metodologías y estrategias que apoyen en su correcta implementación con el fin de resolver las necesidades del usuario final [Booch1998].

1.1 Ingeniería de software.

De acuerdo a la IEEE [SWEBOK2004], la ingeniería de software se refiere a "*la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo y mantenimiento del software, es decir, la aplicación de la ingeniería al software*."

La ingeniería de software proporciona herramientas para administrar de forma eficiente el desarrollo de sistemas de software y se preocupa por la calidad que deben cumplir estos sistemas. Se enfoca en el incremento de la productividad, reducción de costos y mejora de la calidad en el desarrollo continuo de los sistemas de software.

La complejidad de los sistemas hace que frecuentemente no se entreguen dentro del tiempo, costo o calidad requeridos, y existen diversas causas que pueden hacer que falle un proyecto [Kruchten2004]. La ingeniería de software propone un conjunto de prácticas para reducir los riesgos de fracaso de un proyecto. Una de estas prácticas se enfoca en poner un mayor énfasis en el diseño del sistema y se conoce como "*Arquitectura de Software*".

1.2 Arquitectura de software.

La arquitectura de software ha surgido como una parte elemental del proceso de diseño, ésta engloba la estructura (componentes) de los sistemas de software abstrayendo los detalles de implementación, los algoritmos y la representación de los datos. La arquitectura se concentra básicamente en la interacción y comportamiento de los elementos que conforman el sistema de software con el fin de satisfacer los atributos de calidad (portabilidad, mantenibilidad, desempeño, seguridad, etc.). Una arquitectura de

software se desarrolla como el primer paso hacia el diseño de un sistema que contiene un conjunto de propiedades específicas [Bass2003]. En este sentido una arquitectura de software puede ser vista de manera análoga a *"los cimientos"* de la aplicación.

Por otro lado, la arquitectura sirve como medio de comunicación para todos los involucrados en el desarrollo del sistema, es el primer artefacto previo al diseño detallado que se utiliza como una abstracción que puede ser transferida a un nuevo sistema [Bass2003].

El arquitecto de software juega un papel fundamental en el diseño de sistemas, ya que es responsable de proponer la mejor arquitectura, priorizar los elementos que pudieran entrar en conflicto, definir los componentes que representarán la arquitectura y mediar los conflictos que se pudieran generar en la organización por requerimientos contradictorios.

1.3 Líneas de producto.

Uno de los objetivos de la ingeniería de software en el desarrollo de cualquier sistema, es la reutilización de código o componentes de software, con el objetivo de reducir la carga de trabajo, el tiempo de desarrollo y los costos. Una arquitectura de software representa una inversión de tiempo y esfuerzo considerable y es diseñada por expertos; de tal forma que es deseable maximizar los recursos de la inversión tratando de reutilizar ésta en varios sistemas. Éste es el objetivo de una línea de producto de software [Bass2003].

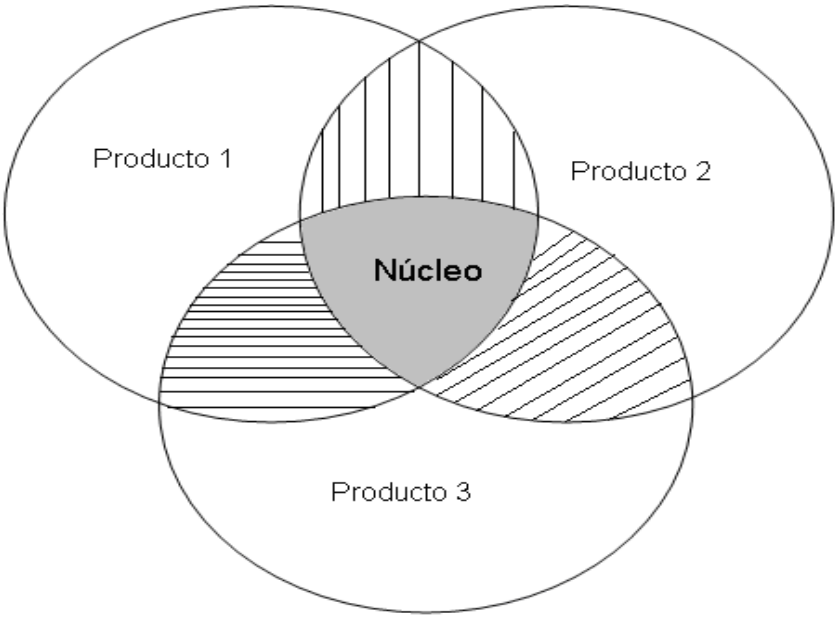

Figura 1: Elementos de una línea de producto.

La idea de reutilizar elementos para generar diversos productos ya es usada de forma exitosa en la industria manufacturera por fabricantes de productos (electrónicos, automotrices), los cuales diseñan un conjunto de elementos base, o núcleo, que posteriormente se reutiliza para crear nuevos productos. La figura 1 representa el hecho de que la intersección de los elementos comunes de los distintos productos caracteriza al núcleo. En el caso particular de la industria de software, el desarrollo de líneas de producto de software requiere de técnicas de ingeniería de software y administración de desarrollo de sistemas, para crear una colección similar de sistemas de software similares entre ellos, y que comparten un núcleo de software común en su producción. En el software, el núcleo común frecuentemente engloba las decisiones arquitecturales. Crear una línea de producto de software exitosa depende de una estrategia coordinada que involucra ingenieros de software, equipos técnicos, la estructura y políticas de la organización en cuestión [Bass2003].

El contexto de este trabajo se enfoca en el desarrollo de un núcleo de arquitectura para línea de producto en un segmento particular, que es el de la informática médica y que se describe a continuación.

1.4 Sistemas de transmisión y almacenamiento de imágenes médicas (PACS: Picture Archiving and Communication Systems).

La evolución de la tecnología de la informática y de las redes de comunicación ha tenido impacto en muchos ámbitos distintos de nuestro mundo actual. Uno de ellos es el de la informática médica, en donde los procedimientos tradicionales de radiología basados en placas radiográficas están siendo reemplazados por medios digitales. Los sistemas PACS ofrecen una alternativa en el manejo de imágenes digitales en forma eficiente a través de dispositivos conectados en una red (ver figura 2), permitiendo proveer servicios de almacenamiento, tratamiento y transferencia de información, para dar soporte a las áreas donde se genera un volumen importante de imágenes [Leotta1993].

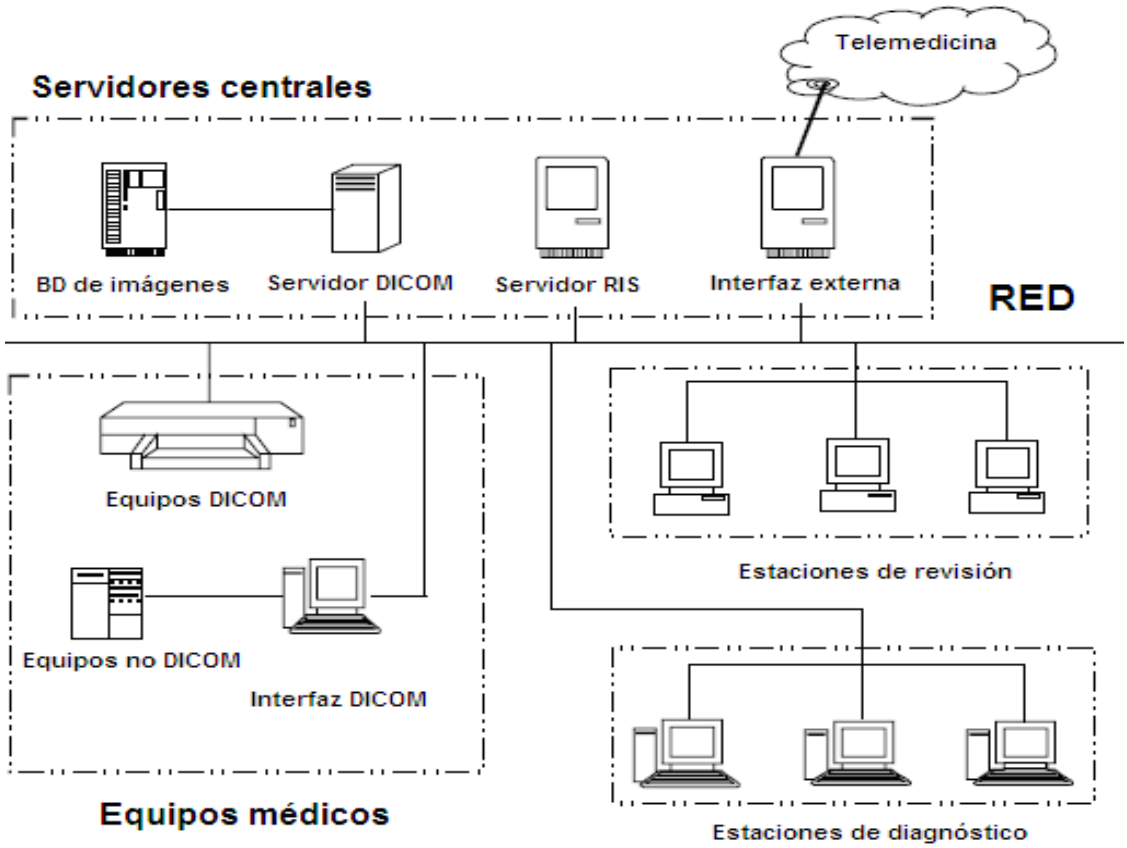

Figura 2: Esquema General de un sistema PACS. Los distintos elementos se comunican y almacenan información mediante un estándar denominado DICOM.

Los PACS pueden incluir desde simples estaciones para realizar consultas, hasta aplicaciones más complejas como: interfaces a equipos que generan las imágenes en áreas de radiología, teleradiología, aplicaciones para almacenamiento, estaciones de visualización, digitalizadores para placas e interfaces con otros sistemas de información (hospitalaria/radiológica).

De manera conceptual, en este trabajo, un PACS es visto como se muestra en la figura 3.

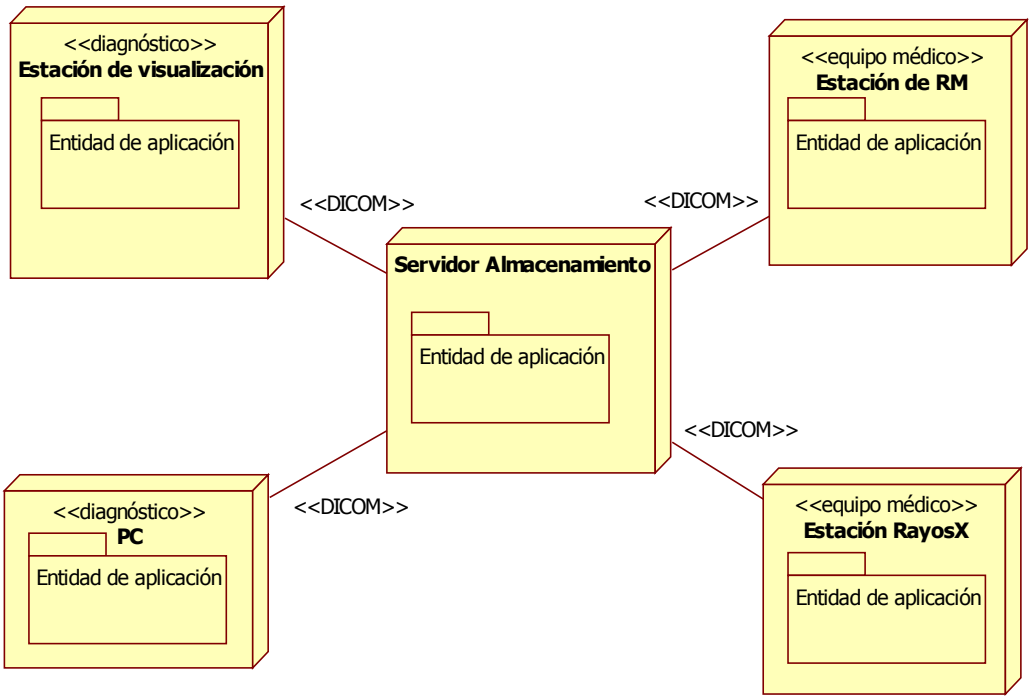

Figura 3: Representación conceptual de un PACS

Como se puede apreciar en la figura, el sistema está compuesto por distintos nodos que albergan software (representado como packages). Dependiendo del tipo de nodo, este software puede estar enfocado a cubrir aspectos de visualización, de almacenamiento o de captura de datos. Los distintos elementos de un PACS comparten, sin embargo, ciertas necesidades. Una de estas necesidades es la de soportar el estándar de comunicación DICOM (Digital Imaging and Communications in Medicine). Este estándar permite que las aplicaciones que se ejecutan en los distintos nodos puedan intercambiar información. El estándar DICOM permite a los nodos comunicarse entre ellos siguiendo un esquema cliente-servidor en el cuál cada nodo puede proveer o requerir servicios específicos, y que varían dependiendo del tipo de aplicación. En la terminología del estándar DICOM, cada aplicación que se ejecuta en un nodo, que soporta un conjunto de servicios DICOM y que participa en un proceso de intercambio de información bajo el estándar, se le denomina "*entidad de aplicación*". En esta misma terminología, el término *conformance* se refiere al conjunto de servicios DICOM que provee o que requiere una entidad de aplicación particular y corresponde a la parte 2 del estándar. Dependiendo del *conformance* de una entidad de aplicación particular, esta última puede proveer o requerir servicios enfocados

a aspectos tales como:

- Consulta de datos.
- Recuperación de datos.
- Comprobación de presencia de otra entidad de aplicación.
- Control de listas de realización de estudios.
- Control de los estados de los estudios (pendiente, diagnosticado, etc.).
- Impresión de estudios.
- Almacenamiento en medios físicos.
- Transporte de información por red.

1.5 Problemática.

La creación de un PACS, tal y como se presenta en la figura 3, requiere del desarrollo de una serie de entidades de aplicación dedicadas a aspectos particulares tales como almacenamiento o visualización. Por otro lado, cada una de esas entidades de aplicación tiene asociado un *conformance* específico. El desarrollo de estas entidades de aplicación tiene entonces varias implicaciones:

- Un desarrollador debe concentrarse tanto en la programación de los aspectos aplicativos, como de la adherencia o compatibilidad al *conformance* requerido.
- El desarrollador debe conocer aspectos específicos de implementación del estándar DICOM.
- La modificación de una entidad de aplicación y por ende su *conformance* puede ser una tarea compleja si el código aplicativo está mezclado con el código de soporte al relacionado con el estándar DICOM.
- Para modificar código de una implementación particular del estándar DICOM, es necesario entender primero las especificaciones del estándar (descritas en 16 tomos o partes) y en segundo término es necesario entender la lógica con la cual fue implementada.
- Las entidades de aplicación comerciales no permiten agregar nuevos servicios DICOM, normalmente, cada uno de éstos se vende como una nueva licencia.

1.6 Propuesta.

Esta tesis parte de la hipótesis de que es posible proveer soluciones que ayuden a resolver la problemática descrita anteriormente mediante la aplicación de un enfoque de líneas de producto. En el contexto de este trabajo, el concepto de "producto" se considera equivalente al de "entidad de aplicación" descrito previamente. Como se puede apreciar en la figura 4, toda entidad de aplicación se puede modelar desde un punto de vista estructural como un sistema construido por capas, en donde la capa inferior contiene, entre otras cosas, los servicios DICOM de acuerdo al *conformance* requerido, y la capa superior contiene la lógica específica de la aplicación. De acuerdo al enfoque de líneas de producto, para este trabajo, el núcleo estaría representando una serie de componentes que se ubican dentro de la capa inferior (ya que la lógica aplicativa varía dependiendo del tipo de entidad de aplicación).

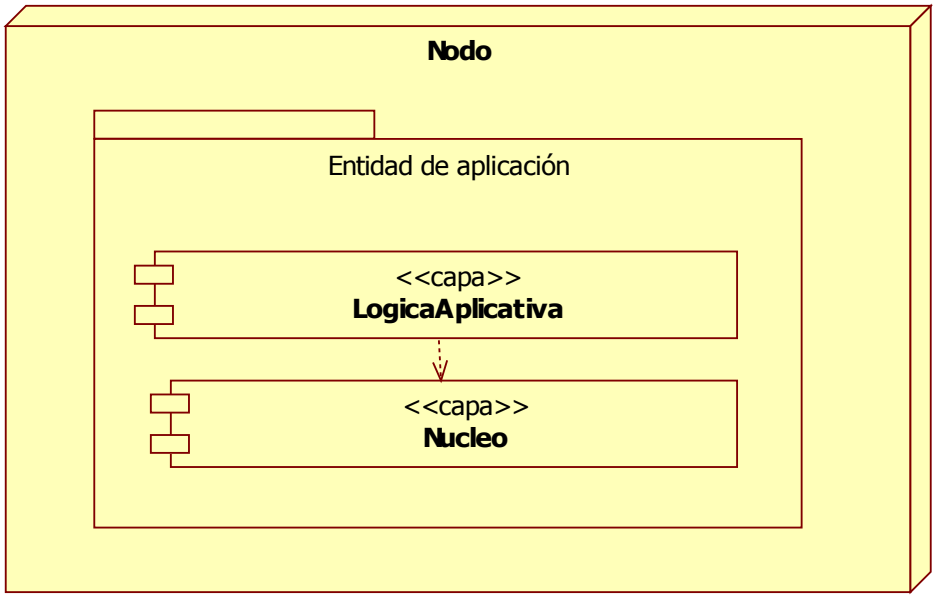

Figura 4: Modelo estructural de capas para una entidad de aplicación.

Uno de los aspectos fundamentales dentro del enfoque de líneas de producto, se refiere al soporte de la variabilidad (mecanismo que permite que los productos, compartan el mismo núcleo, con diferente funcionalidad). En el contexto de este trabajo, ésta variabilidad está relacionada con el *conformance;* dependiendo del tipo de entidad de aplicación, el núcleo arquitectónico debe soportar un número variable de servicios provistos o requeridos.

El desarrollo de una línea de productos debe realizarse de manera disciplinada y siguiendo un enfoque metodológico. En este trabajo, se propone desarrollar la línea de producto en base a la metodología propuesta por el SEI [SEISPLFW2008] que define las siguientes actividades:

- Desarrollo de los elementos de base del núcleo (*Core asset development*). Esta actividad establece los mecanismos o capacidades para construir los productos.
- Desarrollo del Producto (*Product Development*). Esta actividad establece las estrategias para construir los productos.
- Administración de la línea de producto. Esta actividad se encarga de coordinar y supervisar los recursos necesarios para que la línea de producto tenga éxito.

Como parte del desarrollo de los productos, se definen una serie de procesos que son definidos a partir de una técnica de modelado de negocio [IBM2007]. Dentro de esos procesos, se considera una etapa de configuración del núcleo en la cuál el *conformance* se traduce a una descripción que posteriormente es interpretada por un componente del núcleo, este componente se encarga de realizar un ensamblado de implementaciones de servicios DICOM que debe soportar el producto que se está desarrollando.

1.7 Objetivos de la tesis.

En base a lo descrito previamente, se presentan los objetivos de la tesis:

1.7.1 Objetivo general.

El objetivo general de la tesis es realizar el diseño y construcción de un núcleo de arquitectura común para construir productos de PACS.

1.7.2 Objetivos específicos.

Los objetivos específicos del proyecto son los siguientes:

- Definición de una metodología para capturar los requerimientos de los productos y un conjunto de actividades que el desarrollador debe seguir para construir los productos.
- Diseño, documentación e implementación del núcleo arquitectónico.

- Creación de un catálogo de componentes para soportar 4 servicios DICOM para mostrar el uso del núcleo arquitectural.
- Creación de un prototipo de visualizador simple (producto) construido sobre el núcleo.

1.8 Metodología.

La metodología que se siguió dentro de este proyecto fue la siguiente:

- Realización de un estudio del estado del arte, relacionado con arquitecturas de software, líneas de producto y PACS.
- Definición de un proceso para construir los productos de línea de producto para PACS.
- Definición de los requerimientos relativos al núcleo arquitectónico.
- Diseño e implementación del núcleo.
- Evaluación del núcleo a partir de la construcción de un prototipo.

1.9 Alcances y limitaciones.

La construcción de líneas de producto es un tema complejo y extenso. Dadas las limitaciones de tiempo impuestas dentro de este proyecto, este trabajo se enfoca específicamente en el diseño e implementación del núcleo. Dado lo anterior, no se construyen diversos productos basados en este núcleo y únicamente se realiza la evaluación con base a la creación de un prototipo sencillo de visualización. Con respecto a las limitaciones, en el contexto de este trabajo se considera únicamente el soporte de los servicios DICOM para construir un visualizador simple, ya que es un producto que puede mostrar funcionalidad e incluir el núcleo arquitectural, además de funcionar como ejemplo didáctico.

Otra limitación es que de las tres actividades de la metodología del SEI, la de administración relacionada con los aspectos organizacionales no se contempla de forma detallada en este trabajo.

1.10 Estructura de la tesis.

La estructura de la tesis se describe a continuación:

En el capítulo 2 se describen los conceptos importantes de las arquitecturas de software, el impacto de los requerimientos no funcionales en la arquitectura, así como las metodologías de línea de producto y procesos de desarrollo de sistemas. En ésta parte se enfatiza en el estado del arte en arquitecturas y líneas de producto aplicados en el desarrollo de un sistema de información hospitalario para la administración de imágenes médicas.

En el capítulo 3 se describe detalladamente la metodología utilizada para construir el núcleo arquitectural de la línea de producto.

En el capítulo 4 se presenta el núcleo arquitectural de línea de producto propuesto, utilizando un proceso de desarrollo y la metodología de líneas de producto para generar productos para PACS.

En el capítulo 5 se presenta la validación de la propuesta y se describe un conjunto de actividades para generar un prototipo de visualización de imágenes médicas.

El capítulo 6 presenta conclusiones y perspectivas para darle continuación a la conclusión de este trabajo.

2 Arquitectura y líneas de producto de software.

En este capítulo se presentan los conceptos más relevantes para poder comprender este trabajo. Estos conceptos incluyen los temas de arquitectura de software, las metodologías de desarrollo de líneas de producto, los PACS y la importancia del modelado de negocios en la implementación de cualquier sistema de software.

2.1 Arquitecturas de software.

No existe una definición consensual de lo que es la arquitectura de software, sin embargo en este documento dicho término se basará en la definición dada por [Bass2003]: "*La arquitectura de software de un sistema, es la estructura o estructuras del sistema, la cual abarca los componentes de software, las propiedades externas y visibles de estos componentes y las relaciones que existen entre estos*". De esta forma, una arquitectura de software representa la estructura que soporta el funcionamiento de un sistema, también impone reglas de diseño y toma en cuenta como puede cambiar un sistema.

En la definición aparece el término de componente, el cual se refiere a elementos de software con comportamiento e interfaces bien definidas que facilitan el intercambio de información, reutilización y mantenibilidad para la arquitectura.

2.2 Papel de la arquitectura de software en la especificación de un sistema.

El buen diseño de una arquitectura es esencial para lograr componentes de software reutilizables, es decir que se utilicen en varias aplicaciones. En este sentido la arquitectura de software impacta en los siguientes aspectos:

- Define la estructura de un sistema: al diseñar componentes con comportamiento bien específico, el arquitecto define responsabilidades y las tareas más importantes de cada componente. En este sentido, cada componente tiene asignado un rol en la aplicación final. También es importante destacar que se deben minimizar las dependencias entre componentes generando interfaces que faciliten un bajo acoplamiento entre ellos. De esta forma al remplazar algún componente los cambios están bien identificados y las dependencias son mínimas evitando la propagación de errores entre componentes.

- Especifica mecanismos efectivos de comunicación entre componentes: debido a que la arquitectura esta compuesta por múltiples componentes, es necesario crear o utilizar mecanismos de control e intercambio de datos entre los componentes. No obstante, existe un conjunto de patrones o estilos arquitectónicos ya probados y validados en cierto dominio de aplicaciones que utilizan mecanismos de comunicación bien definidos [IanGorton2006]. El arquitecto puede entonces apoyarse en algún estilo arquitectónico predefinido (por ejemplo Cliente-Servidor, el cual define los mecanismos de comunicación entre componentes de forma adecuada) para plantear la arquitectura. También es muy importante destacar que la elección de un estilo arquitectónico particular tiene impacto directo en determinados requerimientos no funcionales.
- Satisface los requerimientos no funcionales (NFR´s *Non Functional Requirements*) de la aplicación final: Este tipo de requerimientos definen como se deben cubrir o alcanzar los requerimientos funcionales de la aplicación final. Estos requerimientos se detallan más adelante.
- Crea un medio efectivo de comunicación entre todos los involucrados: Cuando se identifican los elementos estructurales de la arquitectura (componentes) y las propiedades externas y visibles (comunicación de los componentes, características de desempeño, recursos compartidos, entre otros.) se está en la posibilidad de crear un modelo con diferentes vistas, las cuales especifican diferentes niveles de descripción de la complejidad de la arquitectura. Sin embargo cada involucrado tiene determinado conocimiento del sistema por lo que utilizará la vista que le proporcione la información de su interés de la forma más clara y sencilla.

2.3 Ventaja de la arquitectura de software.

Las ventajas principales que se obtienen al realizar la arquitectura de un sistema son:

- Reducción de los riesgos técnicos y relativos a la calidad asociados a la implementación del sistema: se identifican componentes y patrones arquitectónicos validados y probados los cuales sirven de entrada al diseño del sistema.
- Representa una decisión temprana de diseño: una vez definida la arquitectura, es posible generar prototipos para explorar las consecuencias de las decisiones de

diseño.

- Es una abstracción transferible hacia otros sistemas: se puede reutilizar el diseño de una arquitectura en sistemas que exhiban requerimientos no funcionales similares. De esta forma se reduce el tiempo y los costos para realizar nuevos sistemas.
- Facilita el análisis más preciso de costos y estimación de tiempos: el cálculo de costos y el cálculo de tiempos para los proyectos de software es una actividad muy importante en la construcción de un sistema; ya que permite estimar los recursos necesarios (humanos, tecnológicos, etc.) para la implementación. Al plantear una arquitectura para el sistema se puede conocer mejor la complejidad del sistema y de esta forma estimar los recursos de forma más precisa.

2.3.1 Requerimientos no funcionales.

Como se mencionó anteriormente, la arquitectura de software se enfoca en la cobertura de los requerimientos no funcionales (NFRs) de la aplicación. Los requerimientos no funcionales se asocian a la manera como una aplicación debe cumplir la funcionalidad del usuario final.

De acuerdo con [Wiegers2003], existen 4 categorías principales de requerimientos no funcionales para una aplicación (ver Figura 5):

- Restricciones técnicas: especifican el tipo de tecnologías que la aplicación debe utilizar (plataforma de desarrollo, lenguaje de programación, manejador de base de datos, etc.), normalmente este tipo de NFR no es negociable para la organización.
- Reglas de negocio: especifica reglas bajo las cuales opera una organización.
- Atributos de calidad: especifican requerimientos de la aplicación en términos de escalabilidad, extensibilidad, portabilidad, desempeño, entre otros. En otras palabras un atributo de calidad define como satisfacer los requerimientos funcionales de la aplicación.
- Interfaces externas: representan interfaces que el sistema provee para facilitar la interoperabilidad, con otros sistemas o componentes.

En este trabajo nos enfocaremos en la categoría de atributos de calidad y en la categoría

de interfaces externas, sin perder de vista que las restricciones técnicas y de negocio las impone la organización, debido a lo anterior se propone una metodología para levantar los requerimientos para construir los productos de PACS.

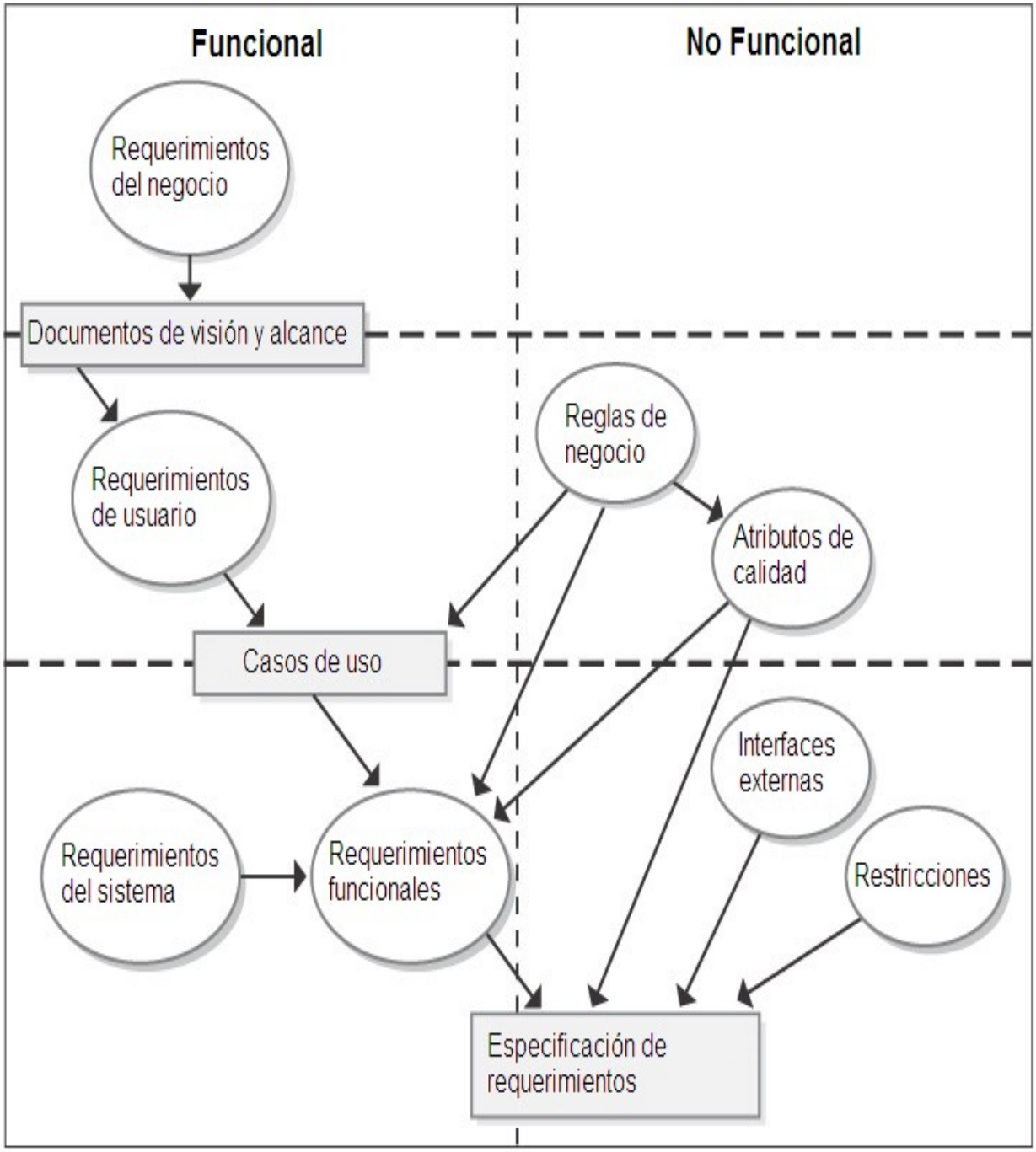

Figura 5: Categorías de requerimientos funcionales y no funcionales dentro del software. [Wiegers 2003]

2.3.2 Atributos de calidad.

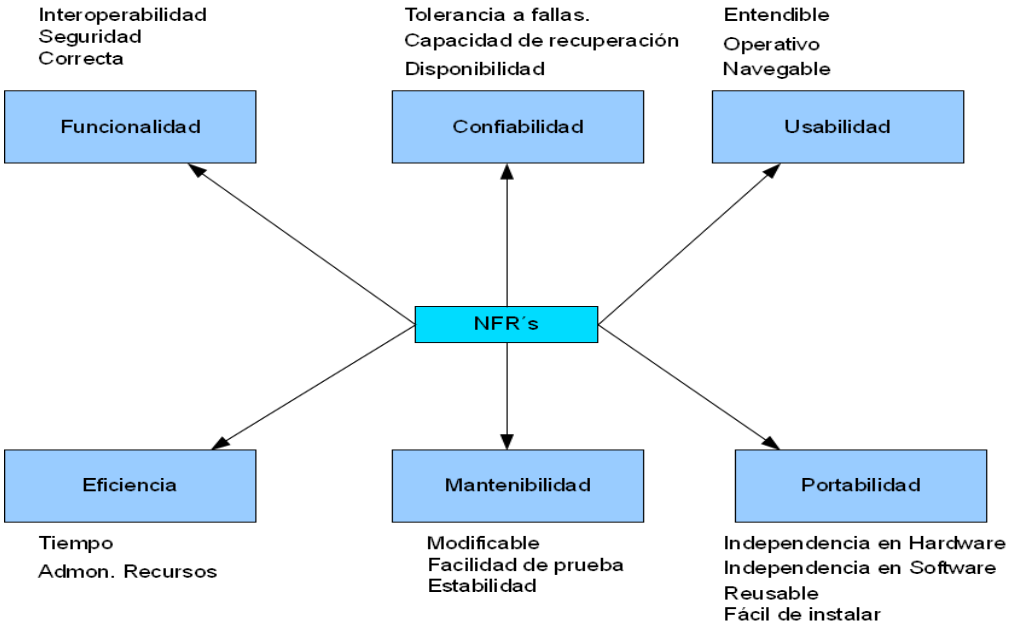

Figura 6: Atributos de calidad del modelo ISO 9126.

Los atributos de calidad son características que permiten verificar y medir el grado de satisfacción de los usuarios y/o diseñadores con respecto al sistema. Dado que la evaluación de la calidad varía de un sistema a otro, existen varias propuestas de modelos de calidad estándar (por ejemplo el estándar ISO 9126) que definen conjuntos de atributos de calidad con el fin de facilitar a las organizaciones el poder realizar la evaluación de aplicaciones específicas [ISO91262007], [IEEE10612007] (Figura 6). Es importante destacar que los estándares consideran una gama amplia de atributos de calidad que pueden seleccionarse para el desarrollo de un sistema, sin embargo el arquitecto debe realizar un análisis para identificar los requerimientos que pudieran entrar en conflicto y seleccionar los adecuados que tengan impacto en el desarrollo del sistema particular.

A continuación se presenta una breve descripción de algunos atributos de calidad comunes para los sistemas de software. Cabe señalar que los atributos de calidad deben ser medibles, sin embargo no existe una métrica única para cada tipo de atributo de calidad. La manera de medirlos debe ser definida como parte del modelo de calidad.

1.- Desempeño (*Performance*): este atributo de calidad, es una medición del tiempo que le

lleva responder al sistema al realizar una función determinada. Evalúa, por ejemplo, que tan eficiente es la implementación de un algoritmo en particular para realizar determinada tarea asociada a un requerimiento de la aplicación.

2.- Confiabilidad (Relia*bility*): este atributo de calidad, es un cálculo del tiempo efectivo dentro del cual el sistema funciona sin errores. Se calcula en base a la longitud de tiempo entre fallas y cuanto tiempo se gasta para restaurar el sistema a su funcionamiento normal. Por ejemplo si el sistema falló un día después de funcionar 20 días de forma ininterrumpida, la confiabilidad del sistema fue: 19/19+1 o 95% disponible.

3.- Funcionalidad (Functionality): este atributo de calidad, es la capacidad del sistema para realizar la tarea para la cual fue diseñado.

4.- Usabilidad (*Usability*): este atributo de calidad, describe que tan fácil es para el usuario final entender y utilizar el sistema.

5.- Seguridad (*Security*): este atributo de calidad, describe la capacidad del sistema para soportar intentos de acceso no autorizados, denegando el ingreso al sistema; es decir solo proporciona servicio a usuarios autorizados, también se enfoca en mecanismos de encripción y transporte seguro de la información.

6.- Modificabilidad (*Modifiability*): este atributo de calidad, es la medición de que tan fácil es modificar el sistema agregando nuevos requerimientos. Los 2 aspectos más importantes para este atributo de calidad son el tiempo y el costo que lleva realizar el cambio.

7.- Portabilidad (*Portability*): este atributo de calidad, mide la dificultad para migrar el sistema a una plataforma diferente para la cual fue implementado. La plataforma puede ser: hardware, sistema operativo, base de datos, etc.

8.- Reusabilidad (*Reusability*): este atributo de calidad, se refiere a la facilidad de reutilizar módulos o componentes del sistema en otros sistemas. Se puede dar en varias formas: plataforma de ejecución, librerías, código fuente, componentes, operaciones, etc.

9.- Integrablilidad (*Integrability*): este atributo de calidad, se refiere a la capacidad del sistema de integrarse o interactuar con otros sistemas. Lo integrable de un sistema depende del uso de estándares de comunicación y del diseño de una API (*Application Programming Interface*) de tal forma que los componentes puedan ser utilizados por otros

sistemas.

10.- Variabilidad (*Variability*): este atributo de calidad, se refiere a como la arquitectura soporta nuevos requerimientos. La variabilidad se puede dar en varias formas las más comunes son: tipos de datos, inclusión u omisión de elementos del núcleo arquitectural, modificación directa de código fuente, etc.

Cabe señalar que mientras la modificabilidad se relaciona con el tiempo y el costo para realizar los cambios, la variabilidad se relaciona directamente con los mecanismos de implementación de los componentes.

2.3.3 Escenarios.

Una técnica de especificación de atributos de calidad es a través de la descripción de escenarios [Bass2003]. Los escenarios permiten describir a los atributos de calidad a través de 6 componentes que son: una fuente de estímulo, un estímulo, una parte de la arquitectura que recibe el estímulo (artefacto), el entorno al momento de recibir el estímulo, una respuesta y una medición de la respuesta (Figura 7).

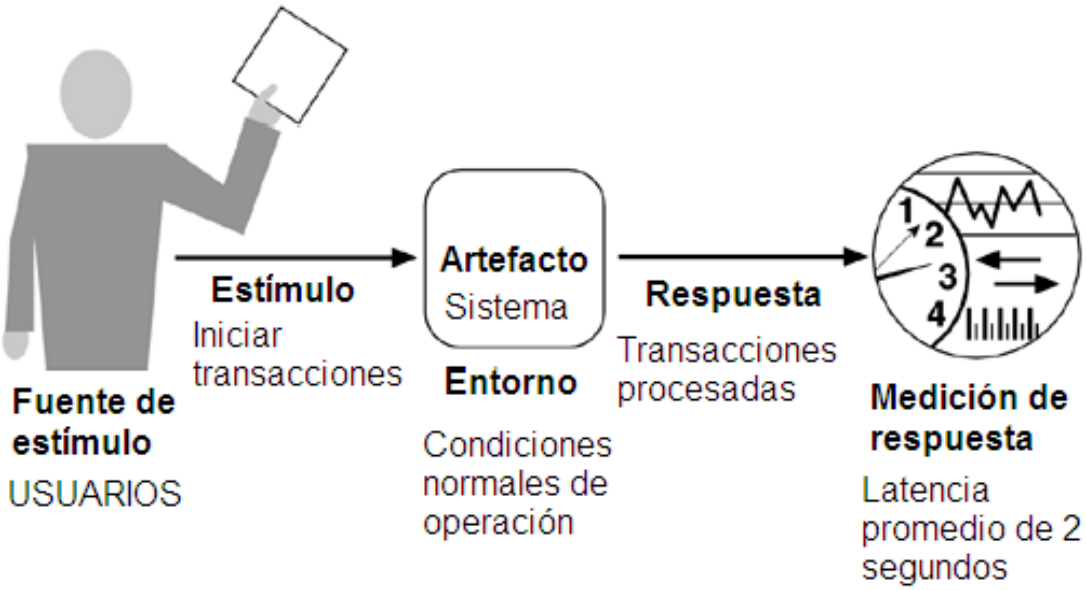

Figura 7: Ejemplo de escenario de atributo de calidad de desempeño. [Bass2006]

2.3.4 Interfaces externas.

Dentro de la categoría de interfaces externas se deben considerar la dependencia entre protocolos de comunicación o considerar el conjunto de mecanismos para la integración de componentes que faciliten la interoperabilidad entre sistemas. Como ejemplo: la

mayoría de los hospitales de tercer nivel, dependientes de la secretaría de salud, cuentan con sistemas de información hospitalaria (HIS por sus siglas en inglés) que no cumplen con alguna especificación estándar para el intercambio de mensajes con PACS o sistemas de información radiológica (RIS por sus siglas en inglés) lo que implica que al realizar una integración HIS-RIS-PACS es necesario desarrollar un conjunto de interfaces de comunicación responsables de traducir los mensajes entre estos sistemas. De lo anterior concluimos que las interfaces externas están relacionadas con el atributo de calidad de integrabilidad.

2.4 Patrones de diseño y arquitectónicos.

El diseño de una arquitectura involucra la resolución de problemáticas asociadas con la satisfacción de los atributos de calidad del sistema. Muchas veces, estas problemáticas y su solución son recurrentes en el contexto de distintos sistemas. Los patrones de diseño y arquitectónicos buscan capturar estas soluciones conceptuales a problemas de diseño recurrentes.

2.4.1 Patrones de diseño.

Los patrones de diseño proponen soluciones de diseño a un nivel de granularidad relativamente fino. Un catálogo muy reconocido de patrones de diseño es el de [Gamma1995]. En este catálogo, se describen patrones enfocados a la creación de objetos, la estructuración y el comportamiento. En el catálogo, los patrones están presentados siguiendo un esquema que se puede dividir en cuatro partes que son:

- Nombre del patrón: identifica a través de un sustantivo el problema, la solución y las consecuencias.
- Problema: explica cuándo aplicar el patrón.
- Solución: proporciona una descripción abstracta (consecuencia) del problema del diseño y cómo se relacionan sus elementos para resolverlo.
- Consecuencias: expresa cuáles son los resultados de aplicar el patrón.

2.4.2 Patrones arquitectónicos.

Los patrones arquitectónicos pueden ser considerados como soluciones de diseño a un

nivel de granularidad superior a los patrones de diseño. Estos patrones se enfocan en la estructuración del sistema y en la satisfacción de atributos de calidad. Al igual que los patrones de diseño, los patrones arquitectónicos están descritos dentro de catálogos, como por ejemplo [Bushmann1996].

Dentro de los patrones arquitectónicos más utilizados se encuentran los siguientes [IanGorton2006]:

2.4.2.1 Cliente Servidor.

En este patrón existen 2 tipos de elementos: cliente y servidor; su coordinación se describe en términos del protocolo que estos establecen para comunicarse por medio de mensajes. Los atributos de calidad que soporta este patrón son: flexibilidad, usabilidad, interoperatibilidad y escalabilidad [SEIDescriptions2008].

En este modelo podemos encontrar 2 tipos de clientes. Clientes ligeros que solo manejan la presentación de datos y la interacción con el usuario y clientes pesados que aparte de la interfaz de usuario; implementan lógica de negocios.

2.4.2.2 3 CAPAS.

Este patrón introduce una separación entre 3 capas: una capa enfocada a la presentación, una capa de servicios dedicada a manejar la lógica de negocio y una capa de datos. Las propiedades específicas de este patrón son las siguientes:

- Sistema modificable: dada la independencia entre las capas de presentación, la lógica de negocios y el almacenamiento.
- Facilita la implantación: todas las capas pueden ejecutarse en la misma máquina o se pueden distribuir en máquinas diferentes.
- Facilita la implementación de aplicaciones WEB.

En este patrón la forma de manejar los atributos de calidad, depende en gran parte de la capa de lógica de negocios, que juega el rol central en la arquitectura. Es muy apropiado utilizarse cuando una aplicación debe soportar un gran número de clientes, peticiones concurrentes y cada petición toma un intervalo de procesamiento.

Como ejemplo, un PACS que soporte almacenamiento en una base de datos puede ser

dividido en las siguientes capas. Capa de presentación serían las estaciones de visualización (consulta o diagnóstico) que pudieran estar ubicadas en diferentes lugares, en la capa de lógica de negocios se encuentran las entidades de aplicación encargadas de resolver las peticiones de los diferentes usuarios de servicio (clientes almacenamiento, consulta-recuperación, listas de trabajo, servicios web, etc.) dentro de un hospital y finalmente en la capa de almacenamiento se encuentra la base de datos que soporta el almacenamiento de los estudios de imagen.

2.4.2.3 Coordinador de procesos.

En la figura 8 se muestran los componentes elementales de este patrón. Este patrón se utiliza comúnmente para modelar los procesos de negocios de una organización, su principal objetivo es la ejecución de dichos procesos y la asignación de tareas a los agentes participantes [IanGorton2006]. Las propiedades más importantes son:

- Encapsulado de procesos: el coordinador de procesos administra la secuencia de procesos necesaria para realizar los procesos de negocios de la organización. El coordinador es un punto centralizado para la definición de los procesos de negocio, lo que lo hace muy flexible para realizar modificaciones.
- Bajo acoplamiento: los componentes que fungen como servidores, no se enteran de su rol en los procesos de negocios, simplemente definen un conjunto de servicios para los procesos de negocios en el que participan.
- Comunicación flexible: la comunicación entre todos los elementos de este patrón puede ser síncrona o asíncrona. En otras palabras, el coordinador espera los mensajes de los servidores y emite una respuesta. También puede implementar un mecanismo de cola para administrar las respuestas de forma tardía o independiente.

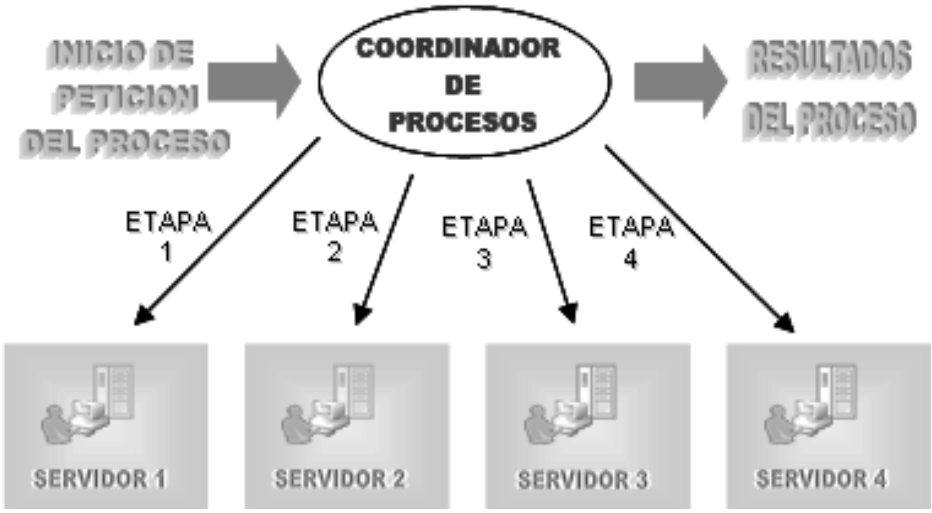

Figura 8: Estructura patrón arquitectónico

Una vez que se tiene uno o más patrones arquitectónicos identificados como candidatos para la arquitectura, es necesario identificar los principales componentes que satisfagan el diseño. También es importante analizar si existen un conjunto de patrones de diseño que faciliten la implementación.

2.5 Representación arquitectural.

Una vez terminado el diseño es necesario documentar adecuadamente la arquitectura de un sistema. Para ello se deben crear varias vistas a través de diagramas. Con esto se puede resumir la gran cantidad de características y componentes que tiene un sistema de software. Estas vistas muestran diferentes aspectos de la arquitectura como son: estructura, comportamiento y la organización de los componentes.

Existen un conjunto de herramientas conocidas como los lenguajes de descripción arquitectural (ADL por sus siglas en inglés), que se utilizan comúnmente para elaborar y representar la arquitectura de software; sin embargo tienen el inconveniente de que se requiere experiencia en su uso e inclusive algunos se utilizan para un dominio en particular. Una alternativa es utilizar un lenguaje de modelado estándar como lo es UML (*Unified Modeling Language*) el cual permite representar varias vistas de un sistema, aprovechando todas sus características para especificar, visualizar y documentar los componentes de una arquitectura [Booch1999].

En [Kruchten1999] se propone un modelo para documentar la arquitectura: esté modelo se divide en 5 vistas establecidas por los casos de uso que tienen impacto directo sobre la

arquitectura. Las vistas se resumen a continuación.

- Escenarios o casos de uso: es la vista principal que enlaza todas las demás. En ella se describen escenarios funcionales que se obtuvieron a partir de la captura de requerimientos.
- Vista lógica: describe los elementos lógicos y sus relaciones necesarias para resolver los distintos escenarios de la vista principal. Se identifican las clases principales para la resolución del sistema.
- Vista de procesos: describe los procesos del sistema y como se comunican entre ellos. Describe aspectos de concurrencia y sincronización del diseño. Se hace énfasis en algunos requerimientos no funcionales (desempeño, disponibilidad, seguridad).
- Vista de implementación: presenta una vista del sistema desde la perspectiva de implementación, compuesta de: componentes, paquetes, subsistemas y librerías. Es recomendable presentar los diagramas de componentes en modelo de capas con un conjunto de interfaces que desacoplen la interacción entre las capas de menor nivel (alta cohesión bajo acoplamiento). Se modela con diagramas de componentes.
- Vista física: ubica los componentes de software en componentes de hardware. Se identifica el software (componentes) que cada nodo (computadora) deberá contener. Refleja aspectos de distribución y se modela con diagramas de implantación.

2.6 Líneas de producto de software.

Uno de los objetivos de la ingeniería de software en el desarrollo de cualquier sistema, es el reutilización a diferentes niveles; en particular reutilización de código o componentes de software con el objetivo de reducir la carga de trabajo, el tiempo de desarrollo y los costos, entre otras cosas. En este sentido, una línea de producto representa una alternativa para la implementación de productos para un segmento particular del mercado. Como se observa en la Figura 1 [página 7], la línea de producto utiliza un núcleo común que soporta variabilidad y cada producto tiene un alcance específico.

2.7 Definición del concepto línea de producto.

En [Bass2003], una línea de producto se define como “un conjunto de sistemas de software que comparten atributos en común, manejan un conjunto de características que satisfacen los requerimientos específicos de un segmento particular del mercado o asociación y se implementan a partir de un núcleo en común.”

Diseñar el núcleo para una línea de producto representa crear la arquitectura de software que responda a los atributos de calidad para cada producto. Una meta es preparar este núcleo para que soporte variabilidad; es decir, es necesario contemplar los aspectos relacionados con las diferencias en la estructura y comportamiento de cada producto en particular. La variabilidad se puede presentar en los siguientes casos:

- Variación de una función (funcional): Una función en particular puede existir en un determinado producto pero no en el resto de la familia o línea de producto.
- Variación en los datos: Una estructura de datos en particular puede variar de un producto a otro. Este tipo de variación en la mayoría de los casos es una consecuencia de la variación en alguna función.
- Variación en el flujo de control: Un patrón de interacción en particular puede variar de un producto a otro.
- Variación en tecnología: La plataforma puede variar de la misma forma que una función; un componente particular puede necesitar un sistema operativo específico o una interfaz de hardware muy concreta.

En otras palabras las líneas de producto de software se refiere a técnicas de ingeniería de software y administración de desarrollo de sistemas, para crear una colección similar de sistemas de software que comparten un núcleo de software común. Esta idea ya ha sido usada en la manufactura de productos (electrónicos, automóviles, etc.), los cuales se crean a partir de un conjunto de elementos base para formar parte de nuevos productos. Crear una línea de producto de software depende de una estrategia coordinada que involucra ingenieros de software, equipos técnicos y la estructura y políticas de la organización en cuestión.

2.7.1 Metodologías de desarrollo de líneas de producto.

Existen varias metodologías para implementar una línea de producto exitosa; sin embargo

no hay una metodología estándar para realizar esta actividad.

El SEI (*Software Engineering Institute*) propone un conjunto de actividades para implementar el núcleo de la línea de producto y los productos que de esta se generan [SEISPLFW2008]. Las actividades importantes que se realizan son: la implementación del núcleo (*core assets),* el desarrollo del producto; ambos coordinados por una actividad de administración que en todo momento se utiliza para controlar la evolución del desarrollo de los productos.

Como se muestra en la figura 9, las 3 actividades mencionadas están estrechamente ligadas y en constante interacción (círculos rotatorios) en donde los artefactos desarrollados en una etapa sirven de entrada para la siguiente y podrían regresar a la etapa anterior (retroalimentación en caso de mejorar determinados aspectos en los artefactos) en caso de identificar modificaciones, antes de pasar a la siguiente etapa.

Figura 9: Proceso de líneas de producto del SEI.

En el capítulo 3 se detalla la metodología de línea de producto, describiendo el conjunto de actividades necesarias para la creación del núcleo arquitectural de este trabajo, con el objetivo de dar seguimiento a la metodología.

2.7.2 Arquitectura de líneas de producto.

De todo el conjunto de elementos del núcleo de la línea de producto, la arquitectura de software representa el elemento central. La arquitectura debe soportar la implementación de todos los productos, considerando cuáles elementos son comunes para todos ellos y cuáles son los mecanismos que soportarán la variabilidad entre productos. En

consecuencia, es necesario pensar en reutilizar la arquitectura en varios productos. Sin embargo se deben identificar un conjunto de mecanismos que soporten variabilidad de los diferentes productos. La dificultad principal reside en cómo hacer que el núcleo se reutilice en diferentes productos con comportamientos diferentes. Dentro de los mecanismos más comunes utilizados para soportar la variabilidad [IanGorton2006] se encuentran los siguientes:

- Variación a nivel de archivo: Existen lenguajes de programación que permiten utilizar compilación condicional y ligado tardío, teniendo impacto directo en variación funcional.
- Variación de diseño: Los lenguajes de programación orientados a objetos permiten diseñar clases utilizando relaciones de herencia, agregación, sobrecarga de métodos, etc. También permiten diseñar clases con interfaces abstractas que pueden ser implementadas con diferentes comportamientos. Estos mecanismos también pueden ser utilizados para soportar variabilidad funcional.
- Variación por el proceso de administración de la configuración de software (SCM): Por medio de este proceso se controlan las versiones de los elementos de la línea de productos. De tal forma que es natural tener varias copias (Ramificaciones temporales) de productos con funcionalidad diferente.

El trabajo que se requiere para implementar una arquitectura de software es considerable, ya que es necesario coordinar recursos (humanos, infraestructura, etc.) así como una inversión para la implementación. No obstante para poder crear una arquitectura de línea de producto es necesario además implementar estrategias de coordinación de ingeniería de software y conocer el negocio para el cual se implementará la arquitectura.

En otras palabras los productos de una línea de producto existen simultáneamente y pueden variar en términos de comportamiento, atributos de calidad, plataforma, configuración, etc. [Bass2003]. En este sentido la documentación de la línea de producto debe mostrar claramente los puntos de variación y como soportarlos.

2.7.3 Beneficios de una línea de producto de software.

El alcance del planteamiento de una línea de producto de software define el conjunto de los productos soportados a partir de los elementos del núcleo. No obstante un buen

diseño de la arquitectura de línea de producto debe soportar la integración de nuevos elementos o productos.

Cuando una organización produce sistemas similares, reutilizando la arquitectura de software, se obtienen los siguientes beneficios:

- Reducción de costos de construcción: debido a la existencia de la arquitectura, el diseño e implementación parten de un núcleo con comportamiento bien identificado. Se requieren menos recursos humanos para construir los productos, teniendo impacto directo en los costos del producto final.
- Reducción del tiempo para integrar los productos al mercado: debido a que los productos se generan a partir de un núcleo común, desarrollar productos es más rápido y en consecuencia se pueden liberar más rápido al mercado.
- Mejora en la calidad de productos: los defectos introducidos en sistemas diseñados de forma independiente deben ser corregidos en cada uno de los sistemas. En una línea de producto al tener bien identificado el núcleo y los componentes de cada producto, los errores se minimizan ya que solo es necesario corregirlos en un artefacto que forma parte del núcleo.

2.8 Modelado de negocios.

El desarrollo de una línea de producto de un dominio requiere de una comprensión detallada del dominio en particular. Una técnica que es útil para lograr dicha comprensión es el modelado de negocios que se describe a continuación.

2.8.1 ¿Qué es el modelado de negocios?

El modelado de negocios, en el desarrollo de sistemas, es el proceso que se emplea para generar una abstracción de la complejidad de un negocio (procesos que se realizan en la organización para el desempeño de las actividades). El objetivo es facilitar y entender el negocio, para poder derivar requerimientos o bien proponer mejoras e innovaciones al negocio.

Para conseguir sus objetivos, una empresa organiza su actividad por medio de un conjunto de procesos de negocio. Cada uno de ellos se caracteriza por una colección de datos que son administrados mediante un conjunto de tareas, en las que un conjunto de

agentes (por ejemplo, trabajadores o departamentos) colaboran de acuerdo a un flujo de trabajo determinado. Además, estos procesos se hallan sujetos a un conjunto de reglas de negocio, que determinan las políticas y la estructura de la información de la empresa. Por tanto, la finalidad del modelado del negocio es describir cada proceso del negocio, especificando [Osterweil1987]: datos, actividades (o tareas), roles (o agentes) y reglas de negocio.

El primer paso del modelado del negocio consiste en capturar los procesos de negocio de la organización. La definición del conjunto de procesos del negocio es una tarea crucial, ya que define los límites del proceso de modelado posterior.

Como primera actividad, se consideran los objetivos estratégicos de la organización. Teniendo en cuenta que estos objetivos generales van a ser muy complejos y de un nivel de abstracción muy alto, y por lo tanto serán descompuestos en un conjunto de objetivos particulares más concretos, que deberán cumplirse para conseguir el objetivo estratégico. Estos objetivos particulares pueden a su vez ser descompuestos en otros, de manera que se defina una jerarquía de objetivos.

2.8.2 Objetivo del modelado de negocios.

Uno de los objetivos del modelado de procesos de negocio es descomponer un proceso a un nivel de detalle que proporcione una visión mas clara para comprender, medir, mejorar y posteriormente ejecutar de manera automatizada el o los procesos de una organización.

Tradicionalmente, el modelado de sistemas de información se ha enfocado en analizar solo los datos y el proceso que se aplica a estos, con el fin de obtener un producto o servicio de información. En consecuencia, este modelo solo se enfoca en una parte de los procesos que una organización administra. Por lo tanto, se requieren nuevas metodologías que contemplen los aspectos de comunicación y coordinación de procesos dentro de una organización.

Por otra parte, el uso de herramientas de software es un elemento fundamental que permiten modelar la descripción de procesos, implantar el modelo y, por último, ejecutar de forma automatizada el proceso o procesos. Estas herramientas utilizan lenguajes que buscan representar un proceso desde un conjunto de vistas que permitan enfocarnos en aspectos funcionales, comportamiento de la organización y manejo de la información. Con

este conjunto de vistas se busca integrar las actividades y productos necesarios para el adecuado modelado del proceso.

También es importante destacar que el lenguaje de modelado unificado (UML), el cual es un estándar en el análisis y diseño de sistemas de software, puede ser utilizado efectivamente para crear los modelos del modelado de negocios [IBM2007].

2.9 Sistema de transmisión y almacenamiento de imágenes médicas (PACS).

Para comprender la complejidad de los PACS, es necesario entender los principales componentes de estos sistemas, la infraestructura y la estructura de la organización en la cual se utiliza este tipo de sistemas. Esto último impacta en los requerimientos ya que impone ciertas restricciones de funcionamiento.

En el desarrollo de un PACS deben considerarse: imágenes obtenidas a través de un sistema de adquisición, sistema de almacenamiento y estaciones de visualización (interpretación, diagnóstico o consulta) integrados todos estos elementos por un sistema de red.

En este punto se presentan los elementos principales de los PACS.

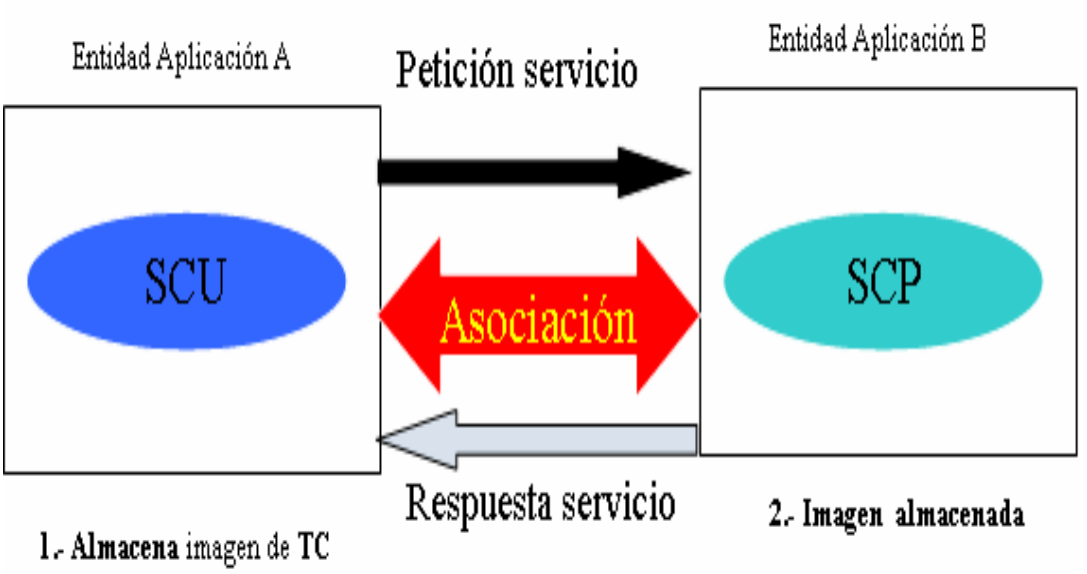

Figura 10: Intercambio de información entre entidades de aplicación DICOM.

2.9.1 Componentes principales.

En el departamento de imagenología de los hospitales, es muy común encontrarse con equipos de varios fabricantes utilizados para la adquisición de imágenes, dichos equipos o modalidades se encargan de generan distintos estudios de imagen: Resonancia Magnética (RM), Medicina Nuclear (MN), Tomografía Computarizada (TC), Radiografía Computarizada (RC), Ultrasonido (US), entre otros. Generalmente, dichos dispositivos pueden inter-operar a través de un estándar llamado DICOM (*Digital Imaging and Communications in Medicine)*, definido desde 1983 por el "*American College of Radiology*" (ACR), representando a la comunidad de radiólogos y la "*National Electrical Manufacturers Association*" (NEMA), representando a la industria. Este estándar representa el resultado de años de esfuerzo para crear un estándar común de intercambio de imágenes médicas digitales [Oleg2008]. El estándar DICOM define un mecanismo de comunicación basado en el patrón arquitectónico cliente/servidor para establecer asociaciones entre dispositivos compatibles, a través del envío de mensajes. Dicho patrón establece un conjunto de reglas para que dos o más entidades de aplicación, como nodos participantes, puedan intercambiar información a través de un conjunto de servicios que se le aplican a un objeto de información bajo una semántica y sintaxis bien definida. Las entidades de aplicación juegan roles de proveedores de servicios (*Service Class Provider* o SCP), usuarios de clase de servicio (Se*rvice Class User* o SCU) o ambos a la vez (ver figura 10). El conjunto de clases de servicios que DICOM soporta se describen a continuación:

- "*Verification Service Class*" (DICOM ECHO): esta clase define un servicio que verifica comunicación a nivel de aplicación (capa aplicación TCP/IP) entre dos entidades de aplicación DICOM. Es importante destacar que todas las entidades de aplicación DICOM se desarrollan sobre la capa de aplicación de TCP/IP ya que facilitan al sistema de software acceder a los servicios de red [Forouzan2004].
- "*Storage Service Class*" (DICOM STORAGE): esta clase de servicio facilita la transferencia de objetos de información (IOD por sus siglas en inglés), permitiendo transportar información entre entidades de aplicación de diferentes tipos (imágenes, reportes, señales, etc.). La información se estructura de acuerdo a las especificaciones de paciente, estudio, serie, e imagen.

- "*Query/Retrieve Service Class*" (DICOM QUERY/RETRIEVE): esta clase de servicio facilita la consulta y recuperación de IOD´s entre entidades de aplicación. Es importante destacar que esta clase de servicio no proporciona un mecanismo generalizado de búsqueda como un lenguaje estructurado de consulta (SQL), ya que utiliza atributos clave comunes a los objetos de información DICOM. En caso de que sea necesario soportar un mecanismo robusto de búsqueda se requiere la implementación de una base de datos.
- "*Print Management Service Class*" (DICOM PRINT): esta clase de servicio permite la impresión de imágenes médicas e información asociada en placas radiográficas o papel.
- "*Media Storage Service Class*" (DICOM MSSC): esta clase de servicio facilita la transferencia de información entre entidades de aplicación a un medio de almacenamiento masivo.
- "*Basic Worklist Management Service Class*" (DICOM WMSC): esta clase de servicio se encarga de administrar las listas de trabajo en el área de radiología. Una lista de trabajo es una estructura para presentar información relacionada a un conjunto de tareas en particular (citas de pacientes, estudios por diagnosticar, etc.).

Es importante destacar que la adecuada implementación del estándar DICOM es necesaria para soportar la comunicación entre PACS [Siegel2002], [Shortliffe2001].

Un PACS generalmente se compone de varios subsistemas que incluyen: equipos generadores de imágenes digitales (modalidades), estaciones de diagnóstico o visualización, sistemas de almacenamiento, sistemas de consulta remota; todos ellos conectados por un sistema de red que permite el intercambio de información (ver figura 4). La arquitectura simplificada de un PACS se puede representar a través de un diagrama de implantación de UML, en donde cada elemento representa un equipo físico, con determinados componentes de software que se comunican entre ellos utilizando protocolos específicos de comunicación tales como DICOM (ver figura 11).

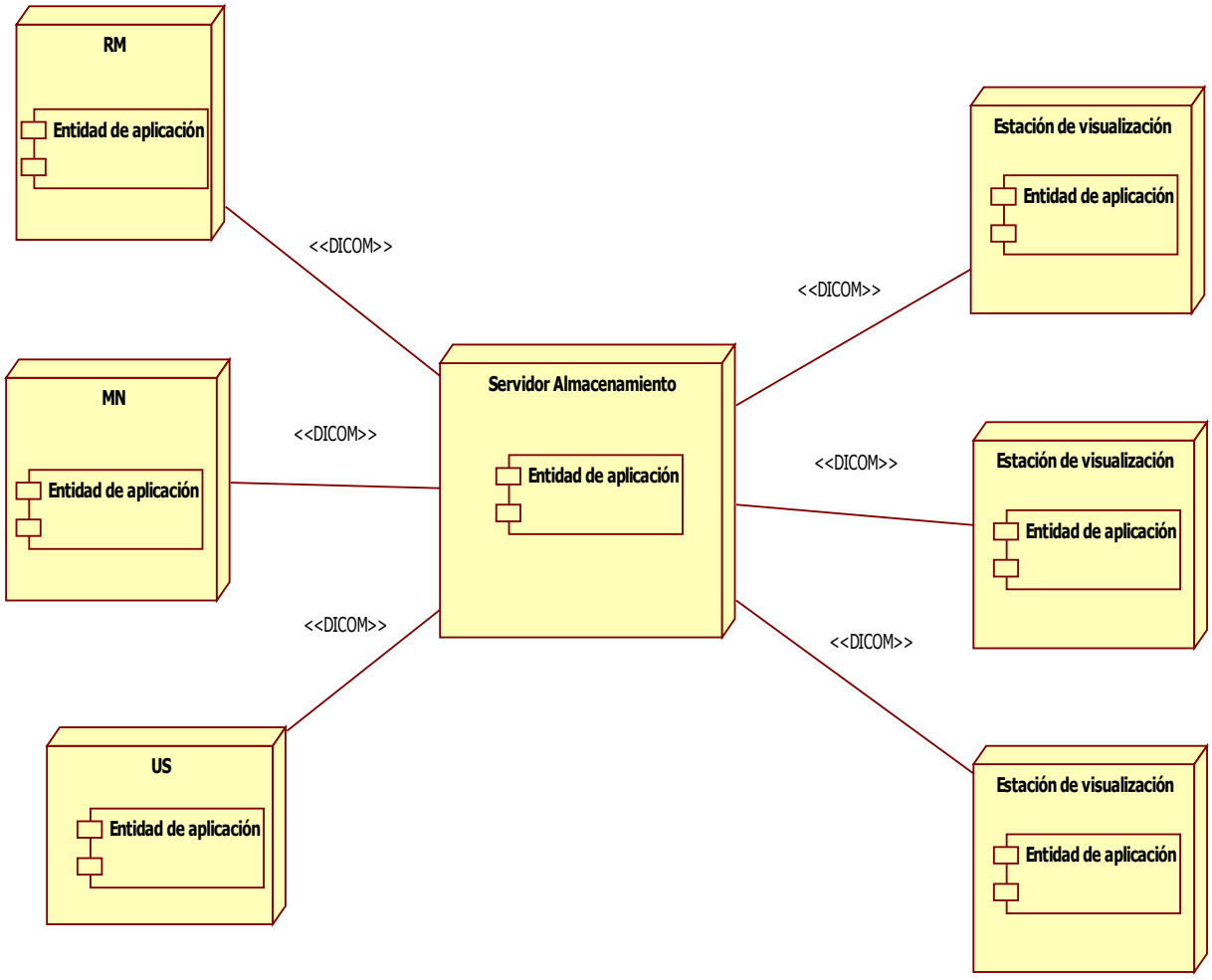

Figura 11: Arquitectura simplificada de un PACS, a la izquierda se presentan distintas modalidades tales como Resonancia Magnética (RM), Medicina Nuclear (MN) y Ultrasonido (US)

A continuación se da una breve explicación de los principales elementos que constituyen un PACS con el objetivo de identificar las restricciones de diseño que estos imponen.

2.9.1.1 Sistema de almacenamiento.

Un elemento fundamental de los PACS, es el sistema de almacenamiento ya que en él se guardan las imágenes recolectadas desde los distintos dispositivos y que posteriormente serán utilizadas para realizar diagnósticos. El sistema de almacenamiento provee servicios DICOM que le permiten recibir peticiones de búsqueda y recuperación de objetos de información DICOM, sus principales componentes son la base de datos y el sistema de archivos que se divide en almacenamiento a corto plazo, almacenamiento a mediano plazo y almacenamiento a largo plazo. Se deben considerar 2 aspectos en el

diseño del sistema de almacenamiento: integridad de datos, cuyo objetivo es evitar la pérdida de datos una vez recibidas las imágenes (se hacen varias copias de la imagen) y eficiencia del sistema, cuyo objetivo es minimizar el tiempo de acceso a las imágenes desde las estaciones de diagnóstico.

Las tareas principales que debe realizar el sistema de almacenamiento se resumen a continuación [HKHuang2004] (ver Figura 12):

- Recibir imágenes desde las modalidades.
- Extraer información demográfica que describe el protocolo de estudio.
- Insertar información en base de datos o sistema de archivos.
- Determinar las estaciones de diagnóstico para cada estudio y enviar los estudios (ruteo automático de estudios).
- Realizar compresión de imágenes, si es necesario.
- Archivar y recuperar imágenes en el sistema de almacenamiento.
- Proporcionar servicios de consulta y recuperación.
- Comunicarse con servidores de aplicaciones que requieran servicios PACS.
- Comunicarse con sistemas de información radiológica (RIS por sus siglas en inglés) para control de flujo de trabajo radiológico.

Este sistema debe soportar cuando menos los siguientes servicios DICOM:

- *DICOM VERIFICATION SCP-SCU*.
- *DICOM STORAGE SCU*.
- *DICOM STORAGE SCP*.
- *DICOM QUERY-RETRIEVE SCP*.
- *DICOM WORKLIST SCP*.

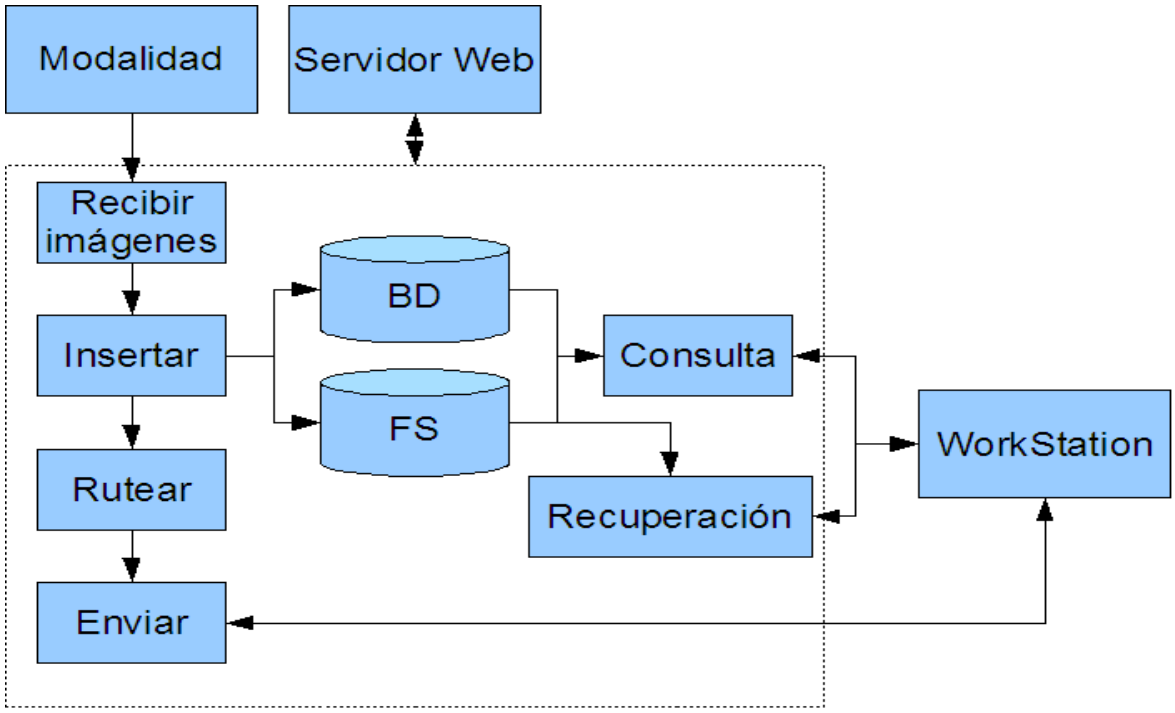

Figura 12: Procesos importantes que se realizan en el sistema de almacenamiento.

Como se observa en la figura 12, el sistema de almacenamiento esta delimitado por un rectángulo con línea punteada; se puede observar que sus responsabilidades son administrar todos los estudios de imágenes generados en las diferentes modalidades y soportar mecanismos automáticos de transferencia de estudios (ruteo) para agilizar el proceso de diagnóstico de los pacientes.

El sistema de almacenamiento también debe ser capaz de proporcionar servicios de recuperación de información para soportar servicios de teleradiológia que se define como la transmisión de imágenes radiográficas de una ubicación geográfica a otra con el propósito de consultarla o interpretarla [Sajeesh2008], esto en la actualidad se está desarrollando cada vez más con el uso de Internet, la mejora en velocidad y ancho de banda de los medios de comunicación y los algoritmos de compresión de imágenes.

2.9.1.2 Sistema de visualización.

Este componente juega un papel muy importante dentro de un PACS y representa tanto a las estaciones de visualización como a las de diagnóstico. Cabe señalar que las estaciones de visualización deben cumplir con normas rigurosas de calidad de despliegue para generar un diagnóstico basado en imágenes digitales. Normalmente estas

estaciones incluyen herramientas de medición (lineal, angular), documentación (texto, integración de reportes), rotación, reconstrucción 3D, agrupamiento de imágenes (modo cine, comparación de series), análisis de regiones anatómicas de interés, manejo de brillo, contraste y niveles de ventana (densidades en diferentes tejidos). La figura 13 presenta un diagrama con la arquitectura de este tipo de sistema.

Los sistemas de visualización deben soportar los siguientes servicios DICOM:

- *DICOM VERIFICATION SCP-SCU*.
- *DICOM STORAGE SCP*.
- *DICOM QUERY-RETRIEVE SCP*.

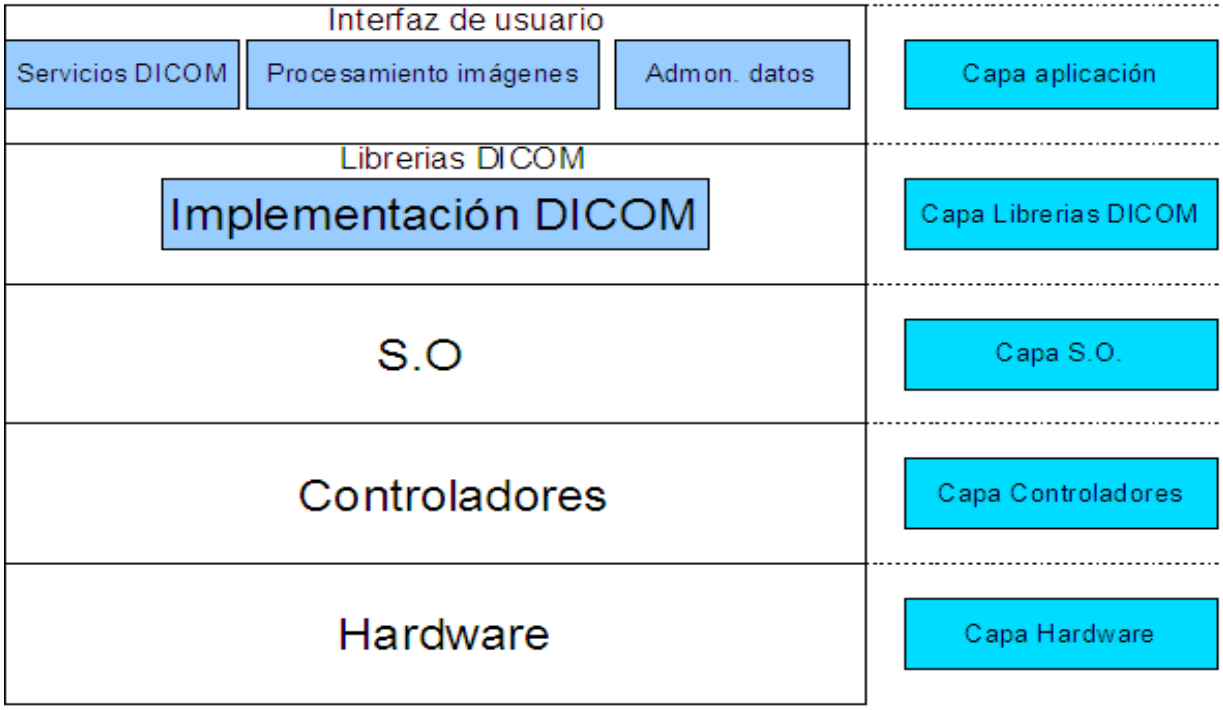

Figura 13: Modelo de capas de estación de visualización. (Los elementos del lado izquierdo ejemplifican cada capa)

La figura anterior muestra que una estación de visualización de imágenes médicas, está compuesta por un conjunto de capas, en donde cada capa contiene determinados elementos, cuyo comportamiento depende del objetivo (diagnóstico o consulta clínica) para el cual será utilizada la estación de visualización. Como ejemplo una estación de diagnóstico requiere de ciertos componentes en hardware como pueden ser mecanismos de calibración de luminiscencia establecidos para diagnóstico. En otro punto podemos mencionar que la capa superior es responsable de soportar la funcionalidad de la

estación, sin perder de vista que es necesario soportar un conjunto de servicios DICOM, soportados por una adecuada implementación del estándar.

2.9.1.3 Sistemas generadores de imágenes médicas.

En el área de radiología de un hospital normalmente existen equipos de diferentes modalidades de imágenes médicas, responsables de generar los estudios demandados por un paciente. Dichos equipos pueden proporcionar la información de imagen en forma digital, lo que facilita la integración con un PACS. Por otro lado las modalidades de rayos X convencional deben ser digitalizadas para poder procesarlas en formato digital. En ocasiones se requiere de interfaces de video para capturar la imagen en forma digital. Debido a lo anterior la cantidad de información generada en un hospital en el área de radiología presenta un gran reto para administrarla de forma eficiente. Como ejemplo la tabla 1 muestra la magnitud de los estudios más comunes.

MODALIDAD	TAMAÑO DE IMAGEN	IMÁGENES POR ESTUDIO	TAMAÑO DE ESTUDIO
Medicina nuclear MN	128×128x12	30-60	1-2MB
Resonancia magnética RM	256×256x12	60-2000	8-500MB
Ultrasonido US	512×512x8(24)	20-230	5-60MB
Tomografía computada TC	512×512x12	40-1000	20-500MB
Radiografía computada RC	2048×2048x12	2	16MB

Tabla 1: Estadística de modalidades más comunes de imagen [IPI2003].

Normalmente los diferentes equipos generadores de imágenes DICOM, cuentan con los siguientes servicios:

- *DICOM VERIFICATION SCP-SCU*.
- *DICOM STORAGE SCU*.
- *DICOM QUERY-RETRIEVE SCU*.

Desde la creación del estándar DICOM en el año de 1998, se definieron las modalidades de imagen como son RM, TC, MN, US. No obstante las modalidades de imagen han aumentado de forma gradual de tal forma que en el año de 2004 se contaban con por lo menos 20 y en consecuencia cada una requiere cierta cantidad de espacio de almacenamiento que impacta directamente en los sistemas de almacenamiento, despliegue y transporte.

A pesar de que los sistemas encargados de generar imágenes son comprados a proveedores comerciales, un PACS instalado en una organización debe soportar la agregación de una nueva modalidad, en este sentido una línea de producto es exitosa ya que al contar con un núcleo arquitectural para construir productos, se pueden agregar nuevas modalidades de forma sencilla debido a la estructura y organización de los componentes que forman los elementos del núcleo. En otras palabras, la creación de una nueva modalidad no debe hacer que un PACS sea obsoleto, el PACS debe ser capaz de soportar la agregación de una nueva modalidad, sin afectar su funcionamiento.

Otro elemento que no es un componente físico de los PACS, pero que es muy importante, es el mecanismo de compatibilidad e interoperabilidad para el intercambio de información. Este elemento es el *conformance* de cada entidad de aplicación y se describe en el siguiente punto.

2.9.2 Niveles de compatibilidad DICOM (*Conformance*).

La parte 2 del estándar DICOM [DICOM2008], describe un documento llamado "c*onformance* DICOM" el cual esta asociado de forma obligatoria a cada *entidad de aplicación.* Una entidad de aplicación se refiere a cualquiera de los sistemas presentados anteriormente. El "*conformance*" proporciona información referente a la interoperabilidad entre entidades de aplicación. En otras palabras; especifica el conjunto de servicios DICOM, los objetos de información soportados y las reglas de asociación que cada entidad de aplicación debe cumplir para intercambiar objetos de información DICOM. El contenido general de un *conformance* se resume en los siguientes puntos:

1.- Descripción general del *conformance*. En este apartado se deben describir los servicios de red y la capacidad para escribir en medios de almacenamiento. Se deben especificar los roles que soporta cada servicio (SCP/SCU) organizados en 4 categorías (Transferencia, Consulta/Recuperación, Administración de flujo de trabajo y administración de impresión) [DICOM2008]. La tabla 2 muestra un ejemplo:

SOP Classes	*User of Service (SCU)*	*Provider of Service (SCP)*
	Transfer	
CT Image Storage	*YES*	*NO*
MR Image Storage	*YES*	*YES*
	Query/Retrieve	
CT Image Storage	*YES*	*NO*
MR Image Storage	*YES*	*YES*
Patient Root Information Model	*OPTION*	*YES*
.....		

Tabla 2: Servicios de red.

2.- Modelo de implementación de las entidades de aplicación locales y remotas. Se realiza un diagrama denominado *diagrama de flujo de datos en la aplicación*, el cual representa el conjunto de actividades y servicios DICOM que las entidades de aplicación soportan (ver Figura 14).

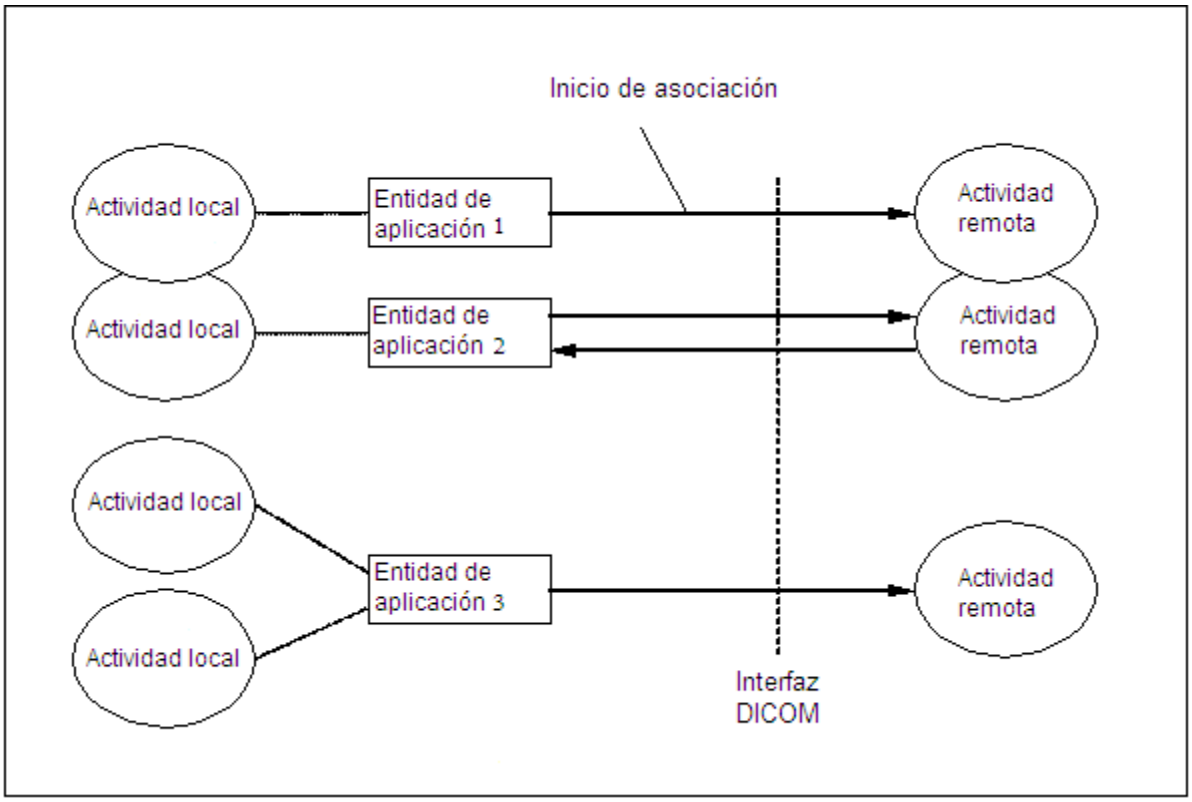

Figura 14: Diagrama de flujo de datos entre entidades de aplicación.

3.- El contexto de aplicación para iniciar y aceptar una asociación de comunicación usada por cada entidad de aplicación.

4.- El conjunto de clases SOP soportadas por cada entidad de aplicación y las políticas de aceptación.

5.- Una descripción de cualquier extensión, especialización o atributos privados que se agreguen a las entidades de aplicación.

Cabe señalar que en el contexto de este proyecto, solo se considera el punto 1, el cual se relaciona con la descripción del *conformance* textual.

2.10 Flujo de trabajo radiológico.

El departamento de radiología y las prácticas radiológicas comparten una doble misión [Keith2006]. Por un lado se enfocan en proporcionar la mayor calidad de atención al paciente y por otro lado buscan maximizar el uso de los recursos hospitalarios. La primera misión requiere de una adecuada selección de técnicas de diagnóstico, personal especializado capacitado y una adecuada coordinación entre los diferentes involucrados en el diagnóstico del paciente. La segunda se logra con el establecimiento de mecanismos de trabajo que coordinen tanto al personal como a la administración de recursos.

Dentro de un centro hospitalario, generalmente se siguen procesos bien definidos a lo largo del tratamiento de un paciente para lograr cumplir las metas expuestas en el apartado anterior. La figura 15 representa el flujo de trabajo radiológico y muestra el conjunto de actividades que se realizan desde el ingreso del paciente hasta la generación de un diagnóstico. El flujo de trabajo se puede resumir de la siguiente manera:

1. Al ingresar el paciente a un hospital, el médico especialista crea o actualiza expediente clínico para determinar el tratamiento que se le dará al padecimiento que el paciente reporta.
2. Posteriormente el médico especialista genera una orden de estudios de rayos X e imagen. En caso de ser necesario, para identificar cual es el mejor tratamiento a su padecimiento.
3. La orden de rayos X e imagen, serán almacenadas en el sistema dedicado a controlar los protocolos de estudio y asignará una fecha para realizar los estudios necesarios al paciente.
4. Posteriormente el paciente se presenta en las áreas respectivas del hospital donde le realizarán los estudios bajo el protocolo determinado en 3.
5. El técnico radiólogo es el responsable de adquirir y almacenar, como primer control de calidad del estudio, los estudios del paciente citado en la modalidad correspondiente.

6. Para mantener disponibilidad adecuada de cada estudio, se requiere que todos los estudios de todas las modalidades sean almacenados en un servidor especializado.

7. En caso que se cuente con la infraestructura necesaria, se pueden soportar servicios de telemedicina por medio de un servidor web.

8. Con el fin de soportar redundancia de la información, se envía una copia del estudio a un sistema de almacenamiento a largo plazo donde se guardará por lo menos durante 5 años, de acuerdo a las normas actuales.

9. Una vez que se almacena la información, el médico radiólogo debe generar la interpretación y/o diagnóstico de los estudios del paciente.

10. El diagnóstico de cada paciente debe ser almacenado en el sistema donde se encuentra su expediente clínico para que el médico especialista lo consulte y determine cual es el tratamiento adecuado al padecimiento que presentó el paciente.

Cabe señalar que generalmente el seguimiento del flujo de trabajo radiológico no se realiza de manera automatizada, algunos hospitales han resuelto este problema a través de la integración de aplicaciones que se ejecutan en paralelo [IHE322006] con el PACS para controlar el proceso radiológico que existe en un hospital. Dichas aplicaciones controlan cada etapa, desde el ingreso del paciente al hospital, hasta que se genera un diagnóstico para darle tratamiento.

Parte de la propuesta de este trabajo incluye la posibilidad de soportar la automatización de la ejecución del flujo radiológico. Dicha ejecución permite administrar, ejecutar y controlar el conjunto de las actividades y procesos que cada participante realiza en la atención al paciente. En este sentido se propone integrar un producto que controle el flujo de trabajo en el proceso de atención al paciente.

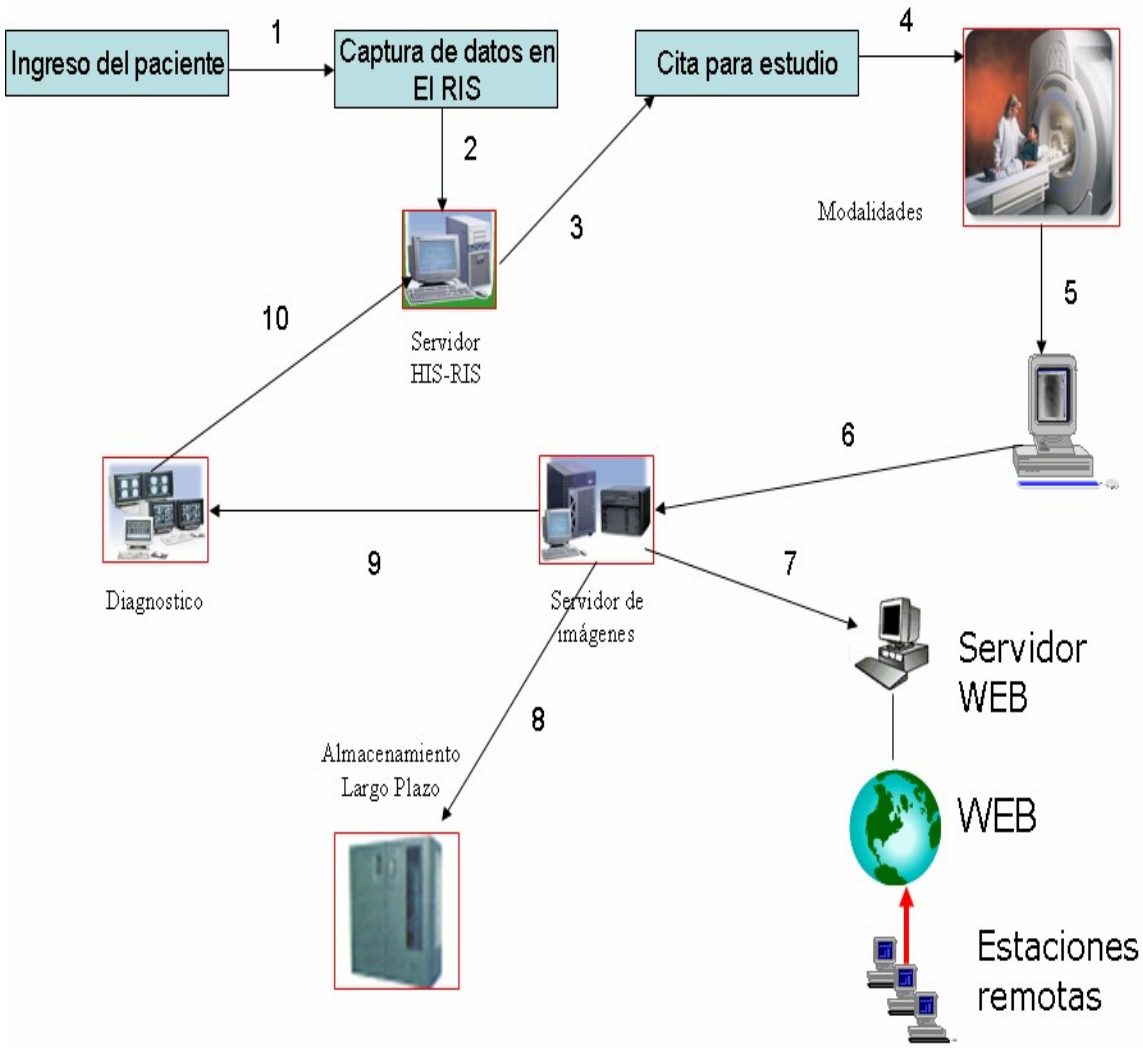

Figura 15: Ejemplo de flujo de trabajo radiológico.

2.11 Síntesis del capítulo.

Una arquitectura de software representa la estructura interna de un sistema, la cual se construye a partir de componentes con comportamiento e interfaces bien definidas. Se enfoca en resolver los requerimientos no funcionales de una aplicación (desempeño, seguridad, extensibilidad, etc.) con el objetivo de soportar la funcionalidad que un usuario final requiere.

Las líneas de producto de software surgieron con el objetivo de aprovechar al máximo los recursos de una organización, reduciendo el costo y los tiempos de desarrollo de software. Una línea de producto busca implementar un conjunto de productos para un segmento en particular del mercado a partir de un núcleo (arquitectura) en común.

El reto para un arquitecto que desea implementar una línea de producto exitosa consiste en preparar el núcleo de tal forma que soporte variabilidad ya sea en comportamiento, estructuras de datos, etc. para todos los productos que se implementarán a partir de la arquitectura. No obstante, es necesario también conocer la organización para la cual se implementará la línea de producto, así como la organización en la que se desarrollará la línea de producto. Debido a esto es muy importante hacer un modelado de negocios que permita conocer la complejidad de la organización ya que la organización define los requerimientos para los productos de software.

3 Propuesta para el desarrollo del núcleo arquitectural.

Como se mencionó en la introducción de este documento, este proyecto se enfoca en el diseño y construcción de un núcleo arquitectónico para construcción de productos que componen a un PACS. Dado que se sigue un enfoque de línea de producto, la construcción de este núcleo se hace siguiendo la metodología propuesta por el SEI [SEISPLFW2008] que se presentó en la sección 2.7.1.

Cabe recordar que dicha metodología define un conjunto de actividades para construir un conjunto de productos de un negocio particular, utilizando una arquitectura construida a partir de componente reutilizables. Las actividades de esta metodología se muestran en la figura 16, que se retoma a continuación:

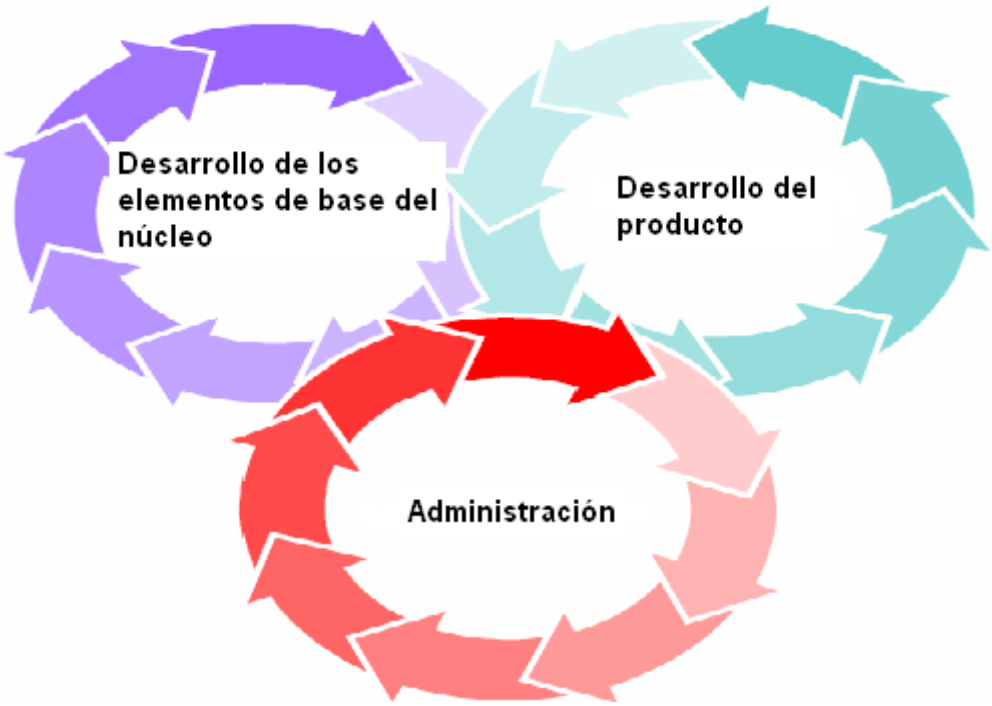

Figura 16: Actividades definidas por la metodología de líneas de producto del SEI.

Este capítulo describe con más detalle las actividades de esta metodología y la manera en que estas actividades fueron realizadas en el contexto de este proyecto.

3.1 Desarrollo de los elementos de base del núcleo (Core Asset Development).

El objetivo de esta actividad es establecer una estrategia efectiva para construir los elementos (o bienes) que forman parte del núcleo de la línea de producto. Como se puede apreciar en la figura 17, esta actividad es iterativa y tiene como entradas los elementos siguientes:

- Restricciones de producto: se refieren a restricciones impuestas a nivel del producto tales como los aspectos comunes, variabilidad, estándares involucrados, entre otros.
- Restricciones de producción: se refieren a restricciones impuestas a nivel de producción relacionadas con quienes son los responsables de construir los productos y que entorno de desarrollo se utilizará.
- Estrategia de producción: se refiere a la estrategia que se plantea para realizar la producción de los distintos productos y que metodología se utilizará.
- Elementos existentes: se refiere a los bienes de los cuáles se dispone previo al desarrollo del núcleo arquitectónico.

A la salida de esta actividad se producen los siguientes elementos:

- Alcance de la línea de producto: se refiere a la descripción de los productos que se pueden construir o que constituyen la línea de producto.
- Base de elementos del núcleo: se refiere a todos los elementos que forman parte de la línea de producto, utilizados para construir algún producto en particular.
- Plan de producción: se refiere a la forma de construir los productos utilizando los elementos del núcleo.

Figura 17: Entradas y salidas de la actividad de desarrollo de los elementos del núcleo.

A continuación se describen estas entradas y salidas en el contexto de este trabajo.

3.1.1 Entradas a la actividad.

A continuación se presentan los artefactos que sirven de entrada para la fase de construcción de los elementos del núcleo.

3.1.1.1 Restricciones de los productos.

Identificar las restricciones de los productos sirve para conocer cuáles son los aspectos en común de los productos, los atributos de calidad, los puntos de variación, los estándares involucrados en la construcción de los productos, así como identificar las interfaces que se utilizaran para comunicarse con otros sistemas.

En el contexto de este proyecto, los distintos sistemas (es decir, los productos) que componen a un PACS (almacenamiento, despliegue, consulta remota, entre otros) comparten elementos comunes y que forman parte del núcleo arquitectural. Los aspectos comunes que comparten los productos, son los estándares de comunicación o

mecanismos de intercambio de datos, entre los que se encuentra DICOM, HTTP, entre otros.

El núcleo de la línea de producto debe satisfacer una serie de requerimientos no funcionales. Los cuales se describen a detalle en el capítulo 4, estos requerimientos se derivan del estudio del estado del arte.

La variabilidad de los productos depende del conjunto de servicios DICOM que cada uno debe soportar, así como la funcionalidad para la cual se desarrolla el producto. En este sentido nos enfocamos en variabilidad a nivel de archivos, relacionada con compilación condicional y variación a nivel de diseño, relacionada con un conjunto de interfaces abstractas que pueden ser implementadas con comportamientos diferentes.

3.1.1.2 Restricciones de producción.

Las restricciones de producción imponen limitaciones a nivel de la producción de los productos específicos. En el contexto de este proyecto, estas restricciones se derivan de la identificación de los responsables para utilizar los elementos del núcleo, las herramientas, así como el proceso de producción.

Los responsables de utilizar los elementos del núcleo, se considera lo siguiente:

- La arquitectura propuesta debe ser utilizada por un desarrollador para construir los productos, la implementación se realizará en lenguaje de programación JAVA, el cual nos permite soportar interoperabilidad ya que la maquina virtual funciona independientemente del sistema operativo.
- Los desarrolladores de los productos deben estudiar y entender la parte 2 del estándar DICOM (*conformance*) para comprender los mecanismos de comunicación entre los productos relacionados con el estándar (Figura 3).

A nivel de las herramientas y con el fin de reducir costos, la restricción principal tiene que ver con el uso de un conjunto de herramientas de desarrollo, librerías y frameworks de software libre. Más específicamente, las herramientas usadas incluyen:

- Spring Framework o simplemente Spring [PRING2008] es una herramienta de fuente abierta que permite desarrollar aplicaciones para la plataforma JAVA

utilizando un mecanismo de inversión de control. Por medio de este mecanismo se especifican respuestas a sucesos o solicitudes de datos concretas, dejando que alguna entidad o arquitectura realice las acciones de control que se requieran en el orden adecuado, para inyectar las dependencias que una aplicación requiera.

- Eclipse [ECLIPSE2007] es un entorno de desarrollo integrado. Utilizado para construir aplicaciones o sistemas de software basado en un modelo de datos estructurado. Desde una especificación de modelo descrita en XML, Eclipse proporciona herramientas para producir un conjunto de clases Java para el modelo requerido. Cuenta con un entrono de desarrollo en modo gráfico, que permite al desarrollador una navegabilidad sencilla en la implementación de aplicaciones. Eclipse permite la incorporación de Apache Ant [ANT2007] como herramienta de compilación y construcción para los proyectos a implementar. Ant tiene la ventaja de estar implementada en Java y no depende de un intérprete de comando o sistema operativo, lo que la convierte en una herramienta portable. Se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas.
- En el contexto de este trabajo se incluirá la implementación del estándar DICOM llamada *Pixelmed* [PIXELMED2007], la cual implementa en Java un conjunto de librerías proporcionadas como fuente abierta, para leer, crear y manipular objetos de información DICOM, ya sea en medios de almacenamiento o transportados por red, sin embargo se contempla en el futuro incorporar implementaciones distintas, como por ejemplo, aquella que se desarrolla dentro de la UAM.

3.1.1.3 Estrategia de producción.

La estrategia de producción se refiere a un enfoque general para realizar tanto el núcleo como los productos.

La figura 18, presenta el modelo estructural de capas de un producto para un PACS, en donde podemos observar la división de funcionalidad (capa lógica aplicativa) y elementos del núcleo, los servicios DICOM que el producto soporta dependen de la configuración que se describe en un *conformance* configurable.

- En este trabajo se propone el diseño de un núcleo arquitectural, estructurado en

diferentes niveles de abstracción o capas, que permiten la integración de componentes (reusabilidad), utilizando un conjunto de patrones de diseño para ensamblar los productos.

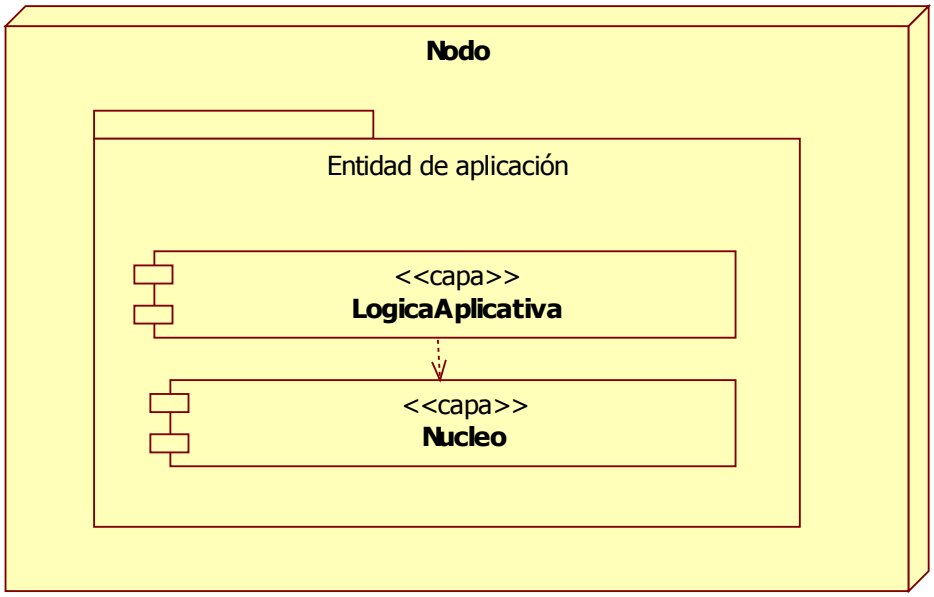

Figura 18: Modelo estructural de capas para una entidad de aplicación, una entidad de aplicación (producto para un PACS).

- Para facilitar el crecimiento de la arquitectura, se define un conjunto de interfaces, que permite la integración o cambio de librerías, con el objetivo de reducir dependencias y facilitar la construcción de los productos (modificabilidad).

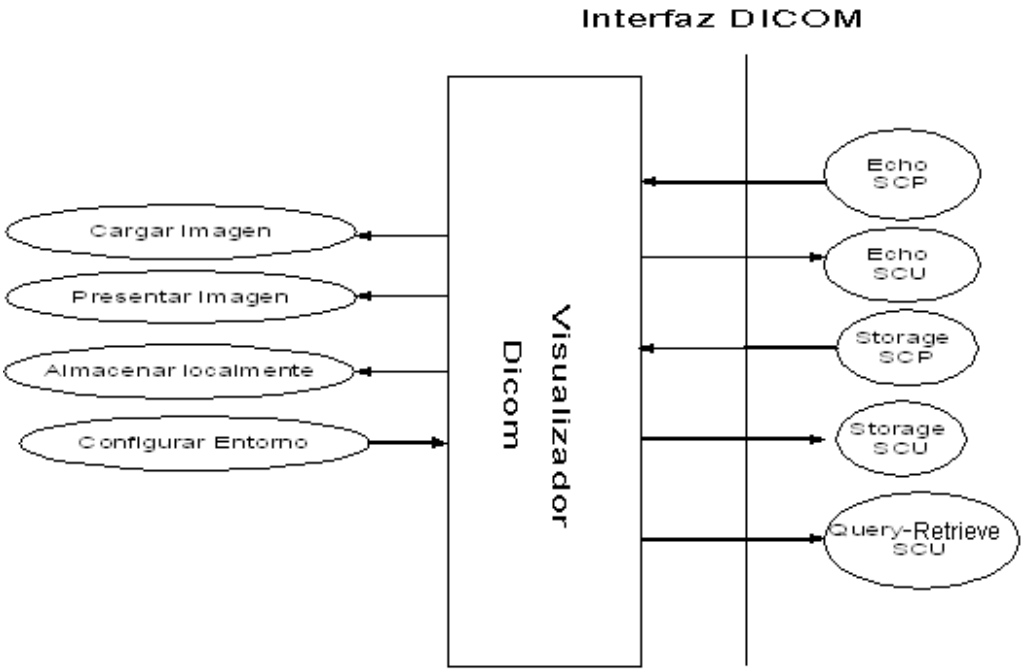

Figura 19: Representación gráfica de un conformance para un visualizador.

- Mapeo del *conformance* textual hacia un *conformance* configurable. Para implementar cualquier producto, es necesario identificar los requerimientos de funcionalidad (capa LogicaAplicativa) y los servicios de comunicación DICOM (capa Núcleo), posteriormente crear una representación gráfica del *conformance* para cada producto. En la figura 19, del lado izquierdo se observan los distintos casos de uso asociados al producto y del lado derecho el conjunto de servicios DICOM que deben soportar a estos casos de uso, es decir su *conformance*. Se define una metodología para capturar los requerimientos de los productos y un conjunto de actividades que el desarrollador debe seguir para construir los productos.

Esta representación gráfica sirve de entrada para escribir un *conformance* configurable en formato XML que se utilizará como descripción de los productos.

3.1.1.4 Elementos pre-existentes.

En el contexto de este trabajo no se considera la existencia de elementos pre-existentes.

3.1.2 Salidas de la actividad.

Al ejecutar la fase de desarrollo del núcleo, se obtiene un conjunto de artefactos, el cual

se presentan en las siguientes subsecciones.

3.1.2.1 Alcance de la línea de producto.

Toda línea de producto debe identificar cuáles son los principales productos que se pueden generar utilizando los aspectos descritos en los puntos anteriores. Es importante recalcar que en el contexto de este proyecto, el concepto de producto se refiere a alguno de los sistemas (entidades de aplicación) que componen a un PACS y cada producto se construye realizando un conjunto de actividades definidas en una metodología propuesta para tal propósito (esta se describe más adelante). En este trabajo se propone que el núcleo arquitectural sea utilizado para construir los siguientes productos:

- Sistemas de visualización de imágenes médicas. Este producto se utiliza para presentar imágenes médicas en formato DICOM.
- Sistemas de almacenamiento. Este producto se utiliza para administrar el conjunto de imágenes médicas que se generan en el área de imagenología de una organización al cuidado de la salud.
- Sistemas de administración de flujo radiológico. Este producto se utiliza para coordinar el conjunto de actividades realizadas por el personal de imagenología, en la atención del cuidado del paciente.
- Para validar el uso del núcleo arquitectural, solo se propone la construcción de un prototipo de visualización de imágenes médicas, el cual incluye un conjunto de servicios DICOM, definidos en un *conformance* configurable, que soporta la variabilidad para los productos.

3.1.2.2 Elementos base de la arquitectura.

Los elementos de base se refieren a los elementos que forman parte del núcleo. En el contexto de este proyecto incluyen las librerías de comunicación y los mecanismos para soportar la variabilidad de los productos (servicios DICOM de acuerdo al tipo de producto).

En el capítulo 4 se realiza la documentación del núcleo arquitectural presentando los elementos que forman la base de la arquitectura.

3.1.2.3 Plan de producción.

El plan de producción describe la manera en que los productos son construidos a partir de los elementos del núcleo. El plan de producción incluye los procesos usados para construir productos y provee detalles para la ejecución y administración del proceso de construcción.

A continuación se presenta un modelo de las actividades que es necesario realizar para desarrollar los productos de un PACS construidos sobre el núcleo común.

En esta propuesta metodológica los procesos serán presentados mediante modelado de negocios usando UML [IBM2007]. En este modelado se emplea el uso de carriles horizontales para representar las fases del proceso; con el objetivo de definir un conjunto de actividades que se deben realizar en la generación de productos para PACS.

3.1.2.3.1 Modelado de actividades.

Este modelo presenta un conjunto de casos de negocio que describe las tareas indispensables para crear los productos.

Una vista general de los casos de negocios se presenta en la figura 20.

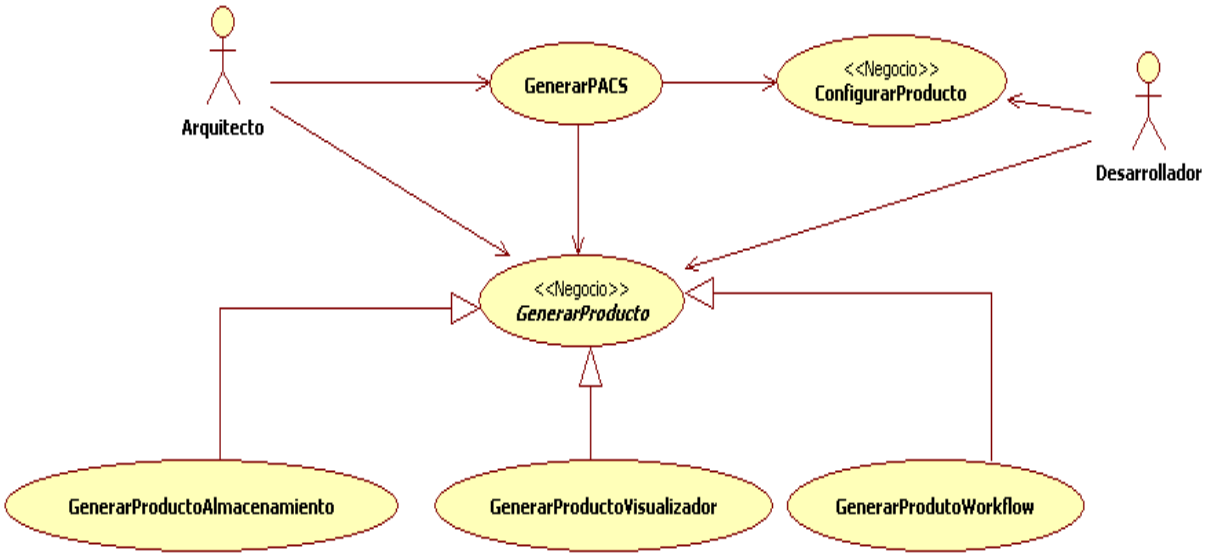

Figura 20: Casos de negocio para construir los productos de un PACS.

Como se puede observar, el caso de negocio más general es GenerarPACS, las actividades que el arquitecto debe realizar en este caso de negocio, son la definición de

todos los productos que se deben crear para un PACS particular, basado en el análisis de la infraestructura de la organización, la producción de estudios de imagen y la organización interna del departamento de imagenología. El arquitecto debe identificar los requerimientos que cada producto debe soportar y coordinar el trabajo del desarrollador en la descripción de cada producto.

Las responsabilidades del desarrollador se relacionan con la identificación de los productos, la generación de los *conformance* configurables (archivo en formato XML) y la implementación de las interfases necesarias para soportar los servicios DICOM que cada producto debe proporcionar.

El arquitecto debe supervisar la generación de los productos para dar seguimiento y conclusión de los artefactos.

En este proyecto se propone utilizar las fases que define RUP (por sus siglas en inglés Rational Unified Process) [RUPDataSheet2007], ya que permiten identificar los elementos que se deben generar en cada una de estas fases para el producto en desarrollo. Además este proceso permite coordinar las actividades que los involucrados realizan para la obtención de los productos.

En las siguientes secciones se detallan las actividades que se deben realizar para cada caso de negocio, en el proceso de generación de PACS a partir de la arquitectura.

3.1.2.3.2 Caso de negocio GenerarPACS.

Los productos que se construyen utilizando la arquitectura serán utilizados en un PACS particular. Por lo que se observa en la figura 21, los involucrados en el desarrollo de los productos, deben identificar los requerimientos de la organización, analizar como satisfacer dichos requerimientos, construir todos los productos que la organización requiera, probar y finalmente instalar cada producto. Todas las actividades se realizan en diferentes fases, cuyo objetivo es facilitar la producción de los productos (plan de producción que se especifica en la metodología de líneas de producto).

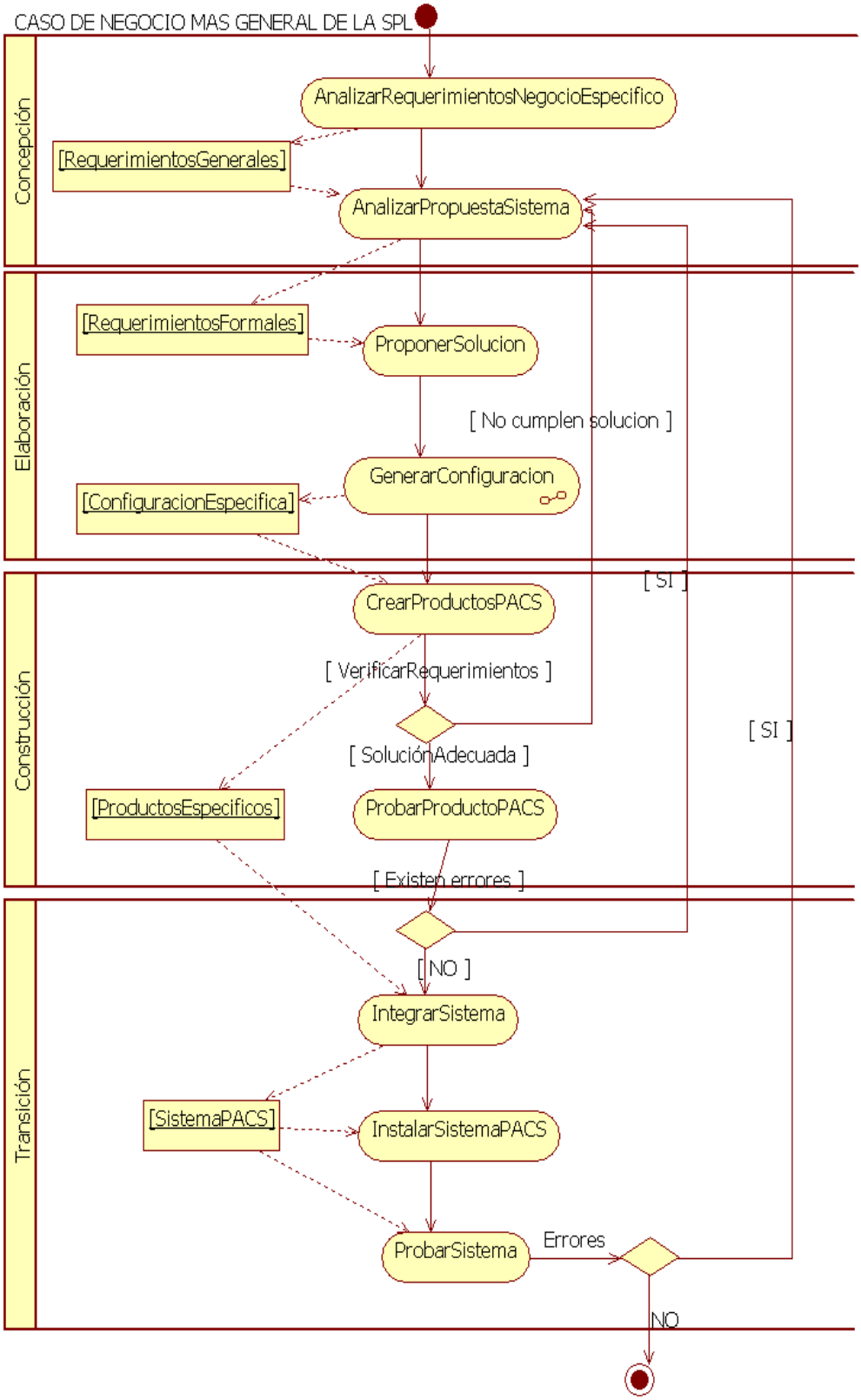

Figura 21: Diagrama de actividades del caso de negocio generar PACS.

3.1.2.3.3 Caso de negocio GenerarProducto.

Este caso de negocio (figura 22) representa la generación de los productos necesarios para integrar un PACS. De esta forma, cada instancia expresa un producto específico.

Es importante observar que en el diagrama se consideran 3 actividades básicas para generar los productos, las cuales se aclaran a continuación:

- Configurar Producto: el desarrollador es responsable de identificar los servicios DICOM que el producto debe soportar y generar el *conformance* configurable.
- Construir Producto: se identifican las interfases que deben implementarse para soportar los servicios DICOM que se definieron en el *conformance* configurable.
- Probar Producto: es necesario crear un conjunto de casos de prueba para asegurar el funcionamiento adecuado de cada producto.

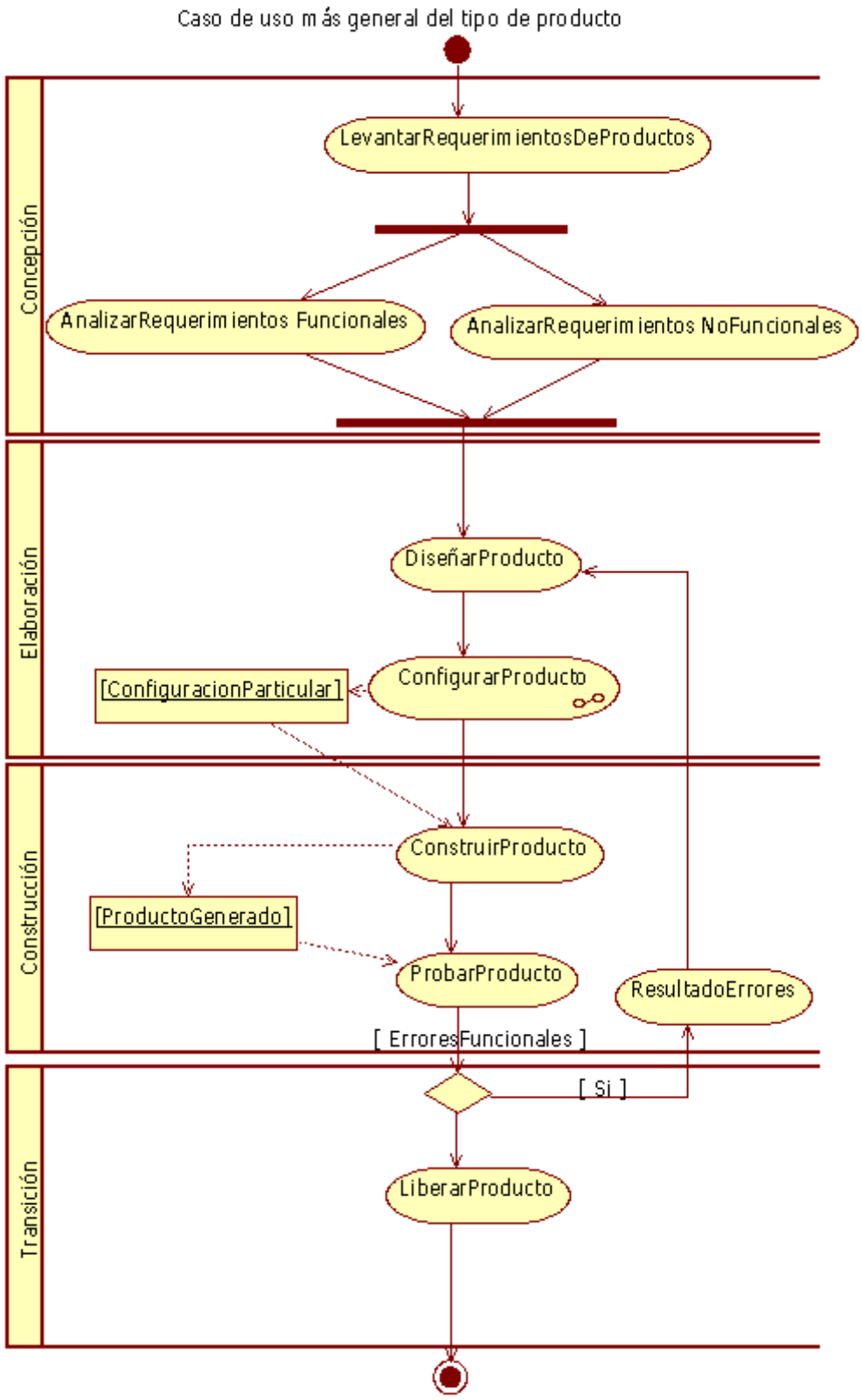

Figura 22: Diagrama de actividades del caso de negocio GenerarProducto.

3.1.2.3.4 Caso de negocio GenerarProductoAlmacenamiento.

Este caso de negocio representa la generación de un producto que soporte almacenamiento para un PACS de los estudios en imágenes (figura 23). En este diagrama se observa que es muy importante hacer un análisis de la infraestructura con la que cuenta la organización, así como la producción de imágenes de esta. El almacenamiento puede ser de 2 tipos, almacenamiento bajo la estructura del sistema de archivos (Paciente->Estudio->Series->Imagen) o almacenamiento en un sistema de base de datos que soporte consultas de búsqueda más eficientes que la primera opción.

Debido a que este diagrama de actividades forma parte de GenerarProducto, el desarrollador previamente definió su *conformance* configurable con los servicios DICOM adecuados (variabilidad del núcleo), así como el conjunto de interfases a implementar (plan de producción de un producto de almacenamiento). El mecanismo de integración de librerías se realizará por medio de un conjunto de patrones de diseño que faciliten la integración de forma automática, de tal manera que el desarrollador se libere de la compilación condicional.

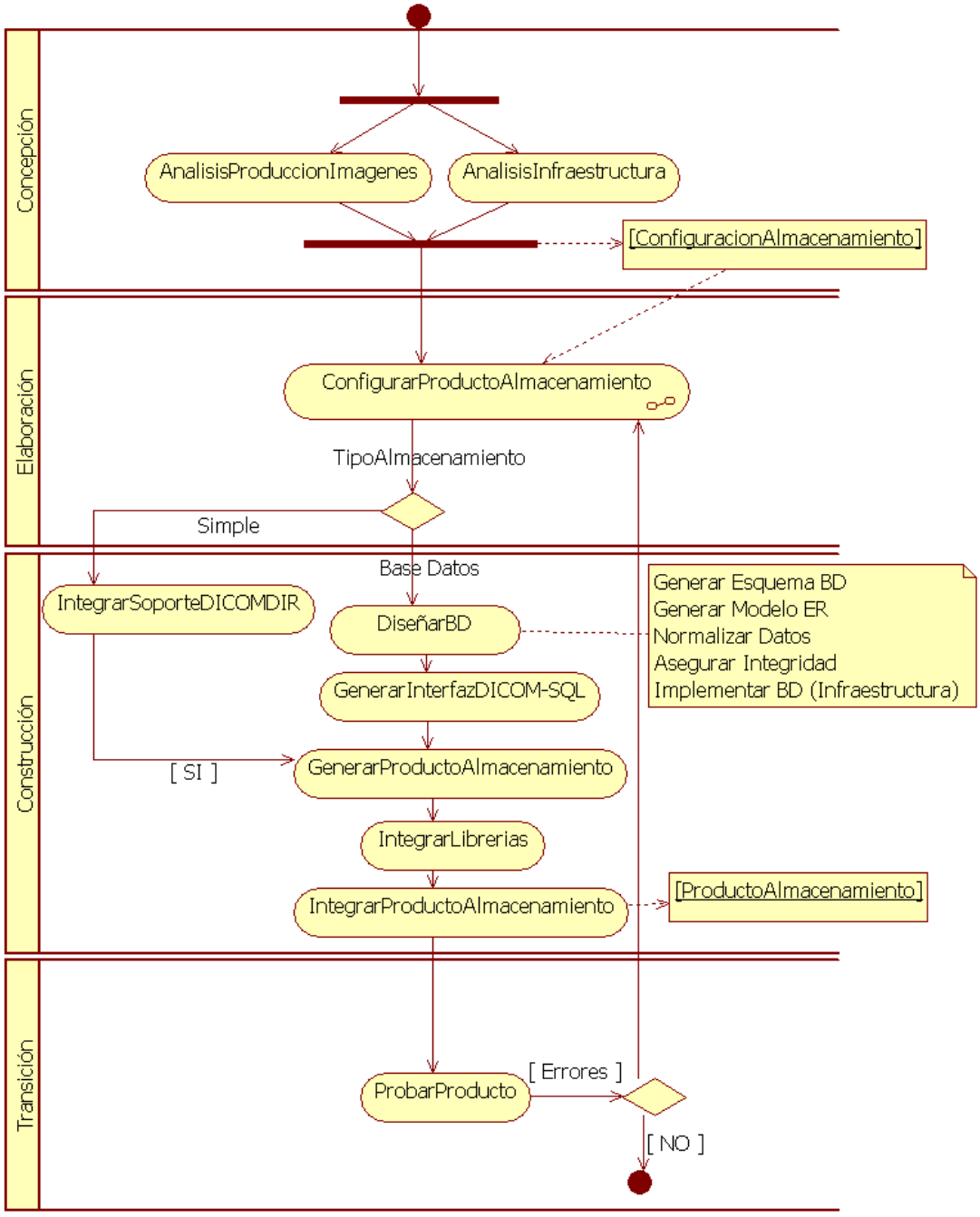

Figura 23: Diagrama de actividades del caso de negocio GenerarProductoAlmacenamiento.

3.1.2.3.5 Caso de negocio GenerarVisualizador.

Este caso de negocio se encarga de construir un producto que presente las imágenes médicas ya sea para diagnóstico o para visualización clínica (figura 24). Un aspecto

importante a considerar es el objetivo para el cual será empleado; ya que si se desea un producto de calidad diagnóstico, se requiere hacer una inversión para adquirir el hardware necesario, con las especificaciones de resolución mínima de 19” en diagonal, 255 niveles en escala de grises y brillo de 60 foot-Lamberts, soporte de calibración DICOM (parte 14 del estándar), se requieren como mínimo 2 monitores para diagnóstico. Esta actividad debe realizarse en el análisis de requerimientos de despliegue, antes de pasar a la configuración del producto.

Por otra parte, para despliegue clínico, es suficiente el uso de un monitor convencional de computadora, permitiendo de esta forma continuar con la generación del producto propuesta anteriormente.

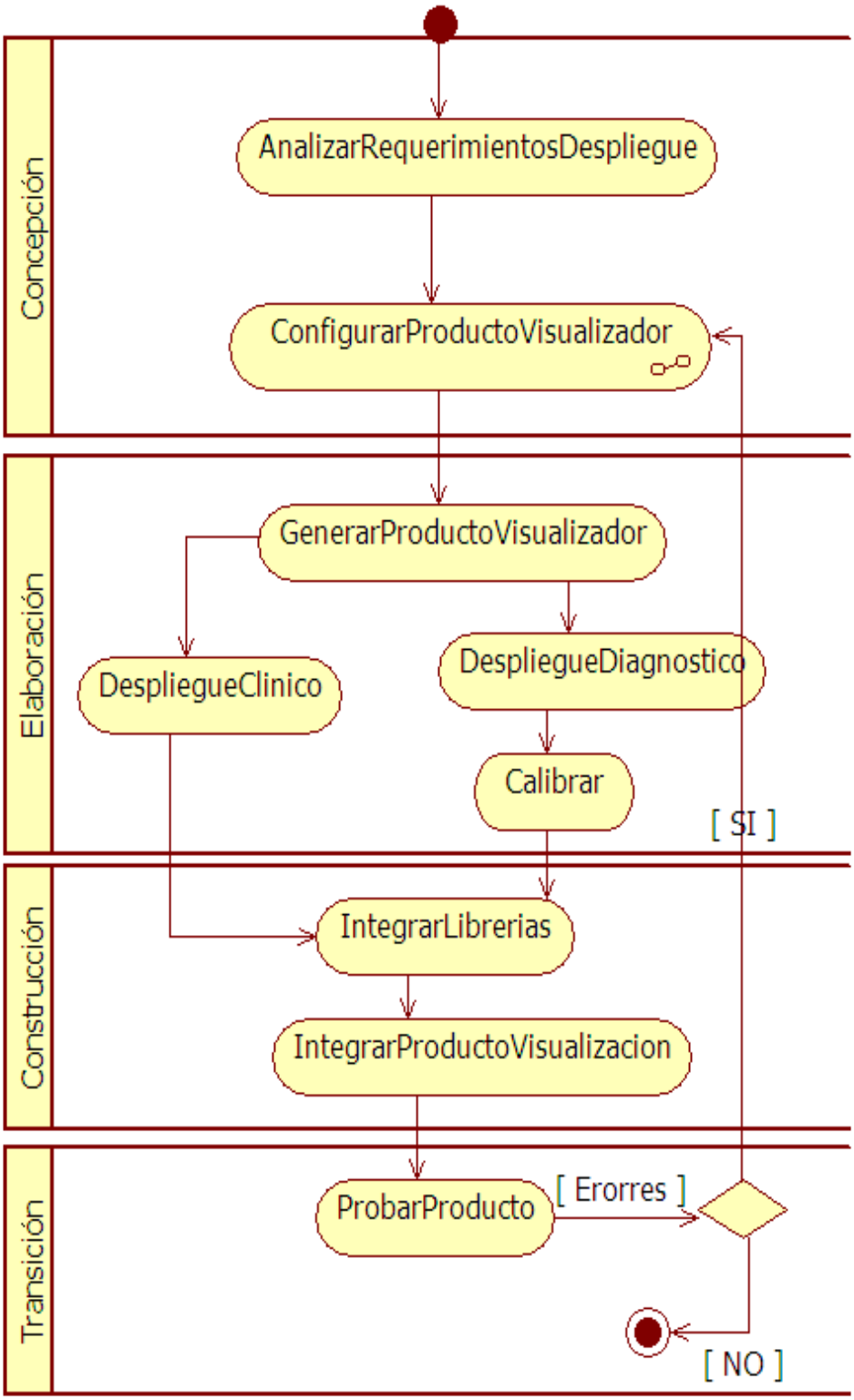

Figura 24: Diagrama de actividades del caso de negocio GenerarProductoVisualizador.

3.1.2.3.6 Caso de negocio GenerarProductoWorkFlow.

Como se mencionó anteriormente, los PACS se han enfocado principalmente en resolver problemas referentes al procesamiento, transferencia de imágenes, comunicación con sistemas de información hospitalaria y los estudios asociados. Sin embargo los aspectos de procesos radiológicos no se han tomado en consideración debido a que las empresas vendedoras de PACS no están involucradas en el entorno hospitalario. En consecuencia se tienen problemas para controlar, coordinar al personal y la cantidad de información que se genera en el área de radiología. Una solución a este problema es la integración de un subsistema adicional que permita coordinar procesos (2.4.2.3) para controlar los agentes, las actividades de cada uno y los productos resultantes. En este sentido se propone la introducción de un producto adicional dedicado específicamente a la administración del flujo de tareas en las áreas de radiología.

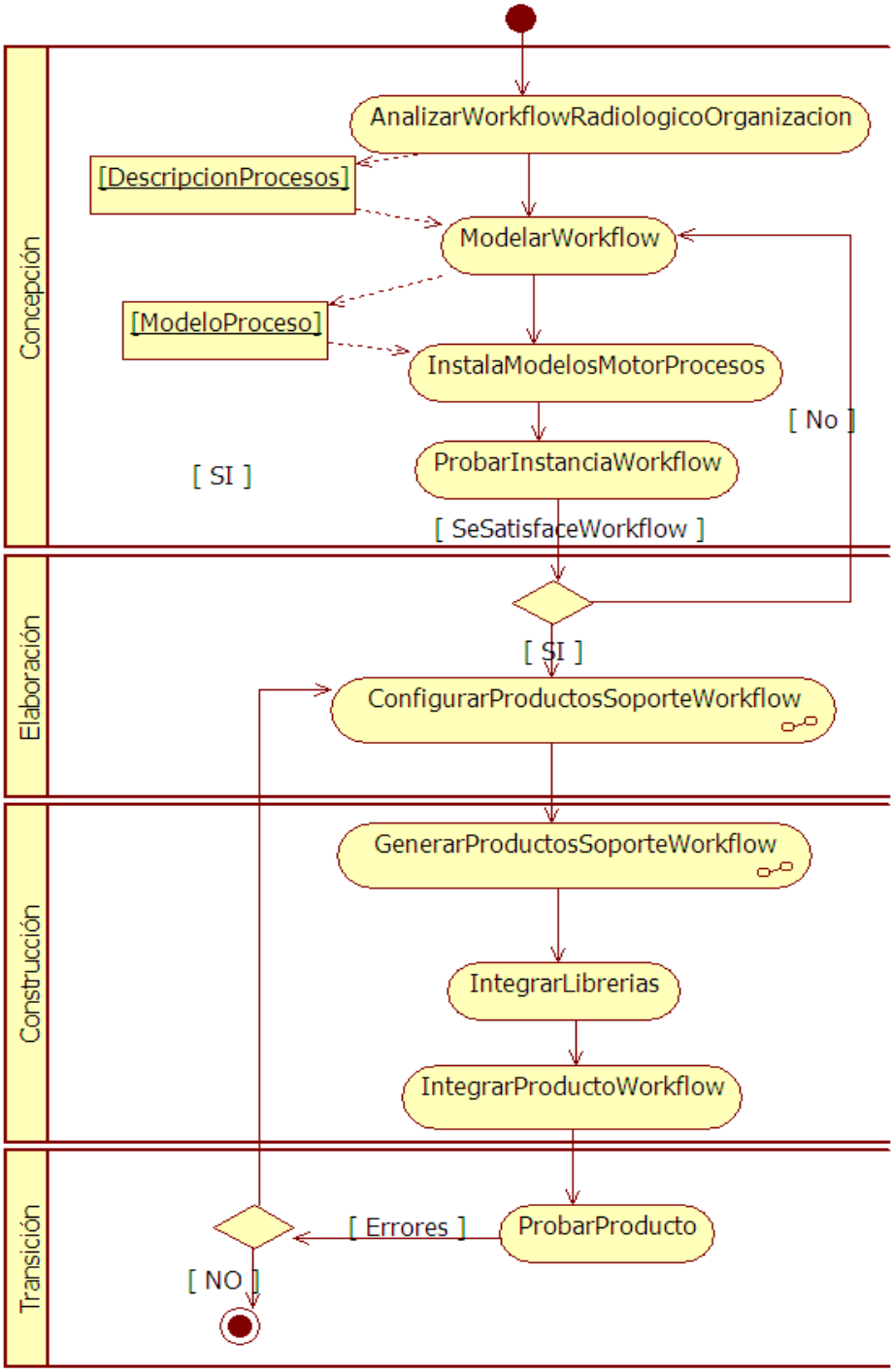

Figura 25: Diagrama de actividades del caso de negocio GenerarProductoWorkFlow

Este caso de negocio se encarga de soportar la construcción de un producto que orqueste las actividades necesarias para administrar el proceso radiológico dentro de un hospital (figura 25). Se desea integrar un componente de administración de procesos y que de esta forma se coordinen las actividades, productos y los roles asignados a todos las

personas participantes en el proceso de radiología. Como se observa en la figura 25, se requiere que exista una descripción de procesos de la organización, para que posteriormente se modele y se instale en un motor de procesos que será el encargado de ejecutar esta instancia. Como podemos observar en los últimos 3 diagramas, todos comparten las actividades definidas de configuración de producto, integración de librerías, integración de producto, esto con el fin de generar un mecanismo automatizado que permita construir cada producto con la mínima intervención del desarrollador.

3.2 Estrategia de desarrollo de productos (Product Development).

La segunda actividad de la metodología es la estrategia de desarrollo de productos. Como se observa en la figura 26, las entradas a esta actividad son el alcance de la línea de producto, los elementos de base del núcleo y el plan de producción. Además de estos elementos (que son salidas de la actividad previa), se tiene una entrada adicional que es una descripción del producto.

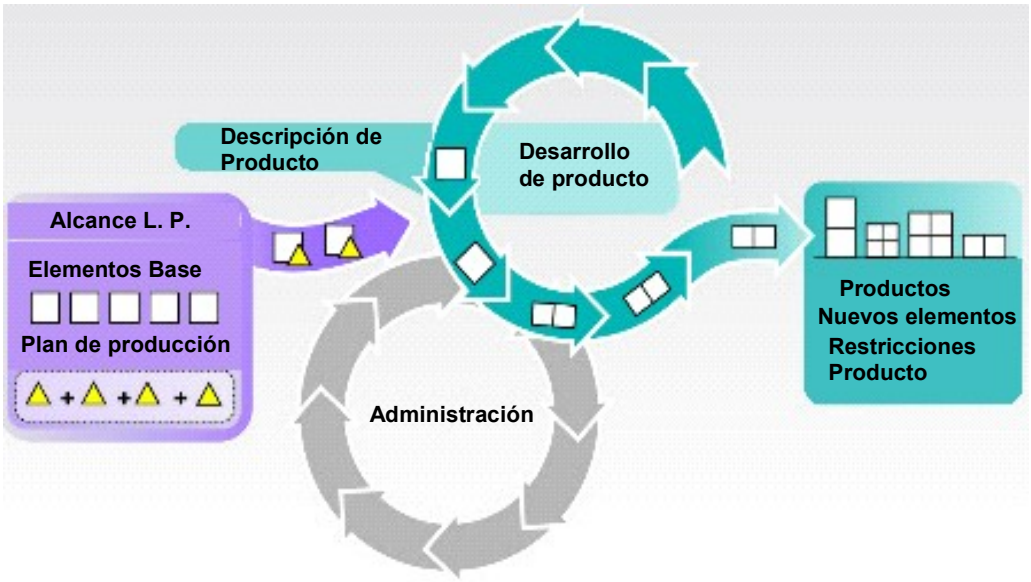

Figura 26: Entradas y salidas de la actividad de desarrollo del producto.

3.2.1 Entradas a la actividad.

La única entrada específica a la actividad es la descripción de producto (ya que las otras entradas son salidas de la actividad previa). En este trabajo, la descripción del producto incluye la lista de servicios DICOM que deben ser soportados por el producto específico (*conformance* configurable). Durante la construcción del producto, y siguiendo el plan de producción, el *conformance* de la descripción es utilizado para configurar el núcleo arquitectural. Dicho *conformance* también es usado para seleccionar los componentes del núcleo que implementan los servicios DICOM a ser incluidos en el producto.

De esta manera el producto final es capaz de soportar los servicios DICOM especificados en su *conformance*. El *conformance* configurable es entonces la abstracción principal que contiene todos los servicios DICOM que el producto final debe soportar.

La descripción del *conformance* se estructura de la siguiente forma:

- Para cada servicio, se genera un identificador asociado a la interfaz de servicio que debe implementar el servicio DICOM.
- Cada servicio debe contemplar un conjunto de propiedades de configuración.
- La estructura del *conformance* configurable se debe escribir en formato XML. En el capítulo 4 se detalla la sintaxis y semántica que representa la descripción de un producto.

3.2.2 Salidas de la actividad.

Como se mencionó anteriormente, la descripción del producto, el alcance de la línea de producto, el núcleo arquitectural y el plan de producción se utilizan en conjunto para obtener un producto funcional y a su vez, si es posible integrar nuevos elementos o bienes a la línea de producto, con el objetivo de mejorarla.

3.3 Administración de la línea de producto.

La actividad de administración juega un papel crucial en la puesta en producción de la línea de producto. La metodología del SEI considera aspectos administrativos a nivel

técnico y organizacional. Por limitaciones de tiempo, los aspectos relacionados con el manejo de recursos y definición de la estructura de la organización no son cubiertos en este proyecto.

3.4 Alcance de la línea de producto.

El presente trabajo utiliza la metodología de líneas de producto del SEI, para diseñar e implementar un núcleo adaptable que permita la generación de los diferentes productos que conforman un PACS. También se define un proceso que permite modelar las actividades para la generación de los productos deseados (sección 3.1.2.3.1).

La validación del núcleo adaptable se realiza a través de la creación de un prototipo de un producto de visualización que se genera a partir de un *conformance* configurable (descrito más adelante). El núcleo integra librerías de fuente abierta que soportan la implementación del estándar DICOM y reducen el tiempo de implementación de los productos.

3.5 Síntesis del capítulo.

En este capítulo se presentó la manera en que se sigue la metodología propuesta por el SEI para líneas de producto en el contexto de este proyecto. Las particularidades de la propuesta presentada incluyen el uso de la técnica de modelado de negocio para la definición de los procesos correspondientes al plan de producción.

La sección siguiente detalla el diseño del núcleo arquitectónico.

4 Arquitectura del núcleo para la línea de producto.

Todos los elementos de los distintos productos en la línea de producto comparten un núcleo común. También, como se mencionó en la sección 2.5, la documentación de la arquitectura es muy importante debido a que representa un conjunto de vistas de un sistema para los diferentes involucrados en un proyecto en particular. Para lograr una línea de producto exitosa es necesario contar con un diseño y documentación adecuada del núcleo (componentes, documentos de apoyo, alcances de la línea de producto, plantillas, etc.) tomando en consideración el alcance y variabilidad de los elementos que se generaran a partir del núcleo de la arquitectura. En este capítulo se presentan los requerimientos y el diseño de la arquitectura de dicho núcleo para línea de producto.

4.1 Requerimientos del núcleo.

El núcleo de la línea de producto debe satisfacer una serie de atributos de calidad que forman parte de los requerimientos del sistema y que se describen a continuación usando la técnica de escenarios.

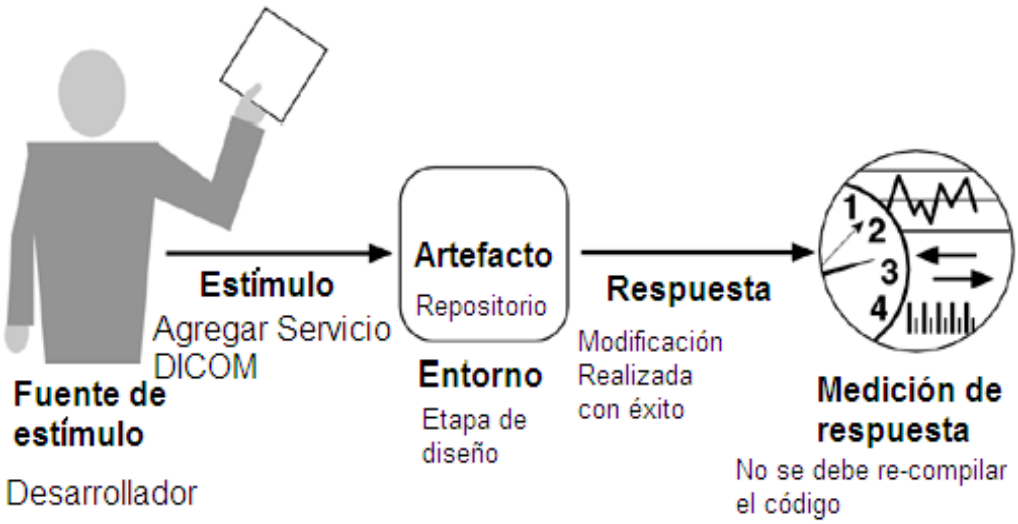

Figura 27: Escenario para agregar un servicio DICOM.

- Modificabilidad / Extensibilidad: La arquitectura del núcleo debe soportar la configuración de los servicios (*conformance*) de un producto particular a través de la composición de componentes reutilizables y sin necesidad de requerir una re-compilación. Esto es en el caso de que no se provea el código fuente de todos los componentes que soportan los servicios DICOM, además no es conveniente tener

un núcleo que incorpore código relativo a todos los servicios DICOM si algunos de ellos no se va a utilizar. Este RNF representa la variabilidad de la línea de producto, ya que cada uno de los productos se diferencía de los demás con respecto a los servicios DICOM que soporta. La figura 27 presenta un escenario de modificabilidad que debe soportar la arquitectura del núcleo.

- Sustituibilidad: La arquitectura debe soportar el intercambio, de forma transparente, de la implementación de los componentes, por ejemplo, para realizar una sustitución de los componentes que implementan el estándar DICOM. La figura 28 muestra un escenario de sustituibilidad.

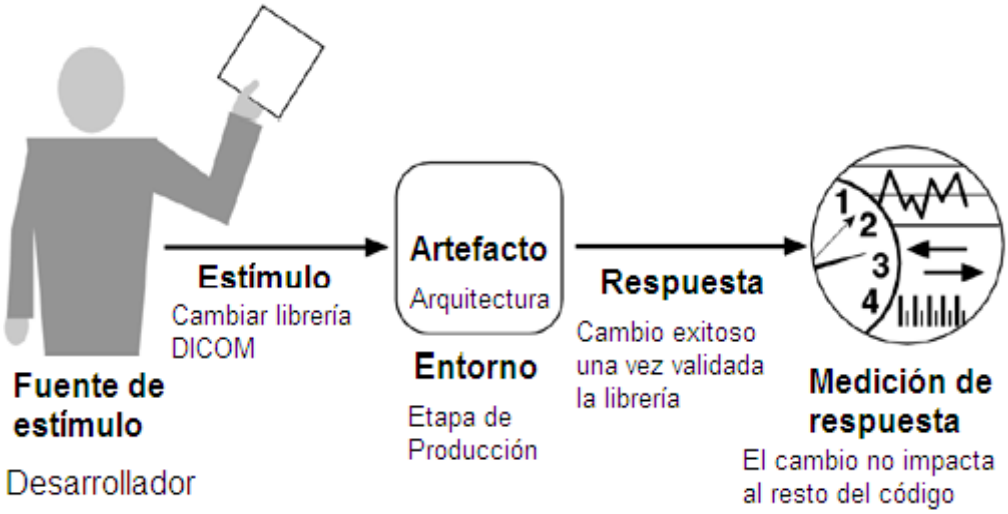

Figura 28: Escenario para cambiar librerías DICOM.

- Portabilidad. Dada la heterogeneidad de los equipos en los entornos hospitalarios, los distintos productos deben poder ejecutarse en diversos tipos de hardware. Debido a lo anterior la arquitectura del núcleo debe ser independiente de sistema operativo y el hardware en el que se implementa. La figura 29 muestra un escenario de portabilidad.

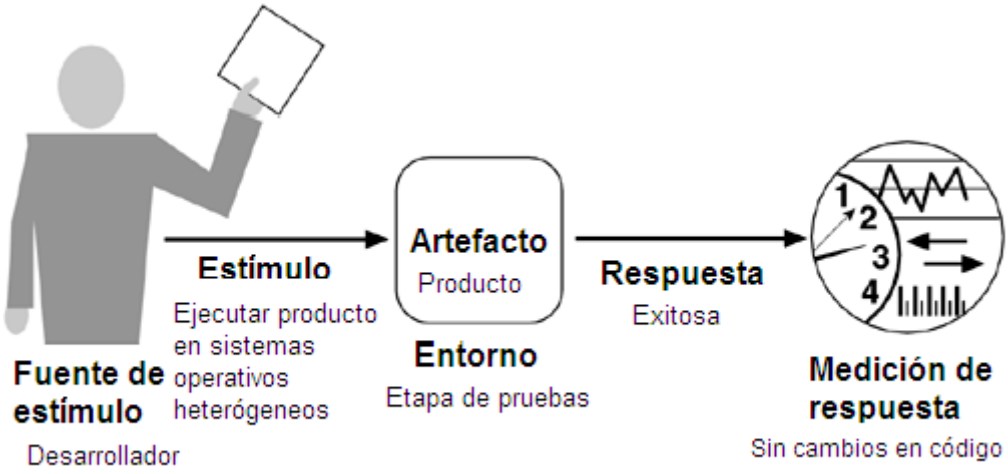

Figura 29: Escenario para cumplir con la portabilidad.

4.2 Diseño del núcleo.

A continuación se describen diversos aspectos relacionados con el diseño del núcleo incluyendo una descripción de los componentes y las interfaces que lo conforman, la manera en que se configuran los servicios de un núcleo particular y el repositorio de componentes.

4.2.1 Componentes de soporte de servicios DICOM.

Como se mencionó en la sección 3.1.1.3 de definición de los elementos del núcleo, cualquier producto (entidad de aplicación) se estructura en 2 capas (capa del núcleo y lógica aplicativa), lo que facilita la identificación de componentes y comportamiento.

Dentro de la capa del núcleo se encuentran las librerías de comunicación DICOM y el conjunto de componentes que implementan los servicios soportados por un producto particular (esto con el fin de soportar la variabilidad). La figura 30 representa lo anterior, en ella se puede apreciar una configuración particular del núcleo que incluye tres componentes que son AlmacenamientoSCU y AlmacenamientoSCP así como VerificacionSCU. Cabe señalar que los diversos componentes que implementan servicios DICOM específicos dependen de la librería de comunicación DICOM.

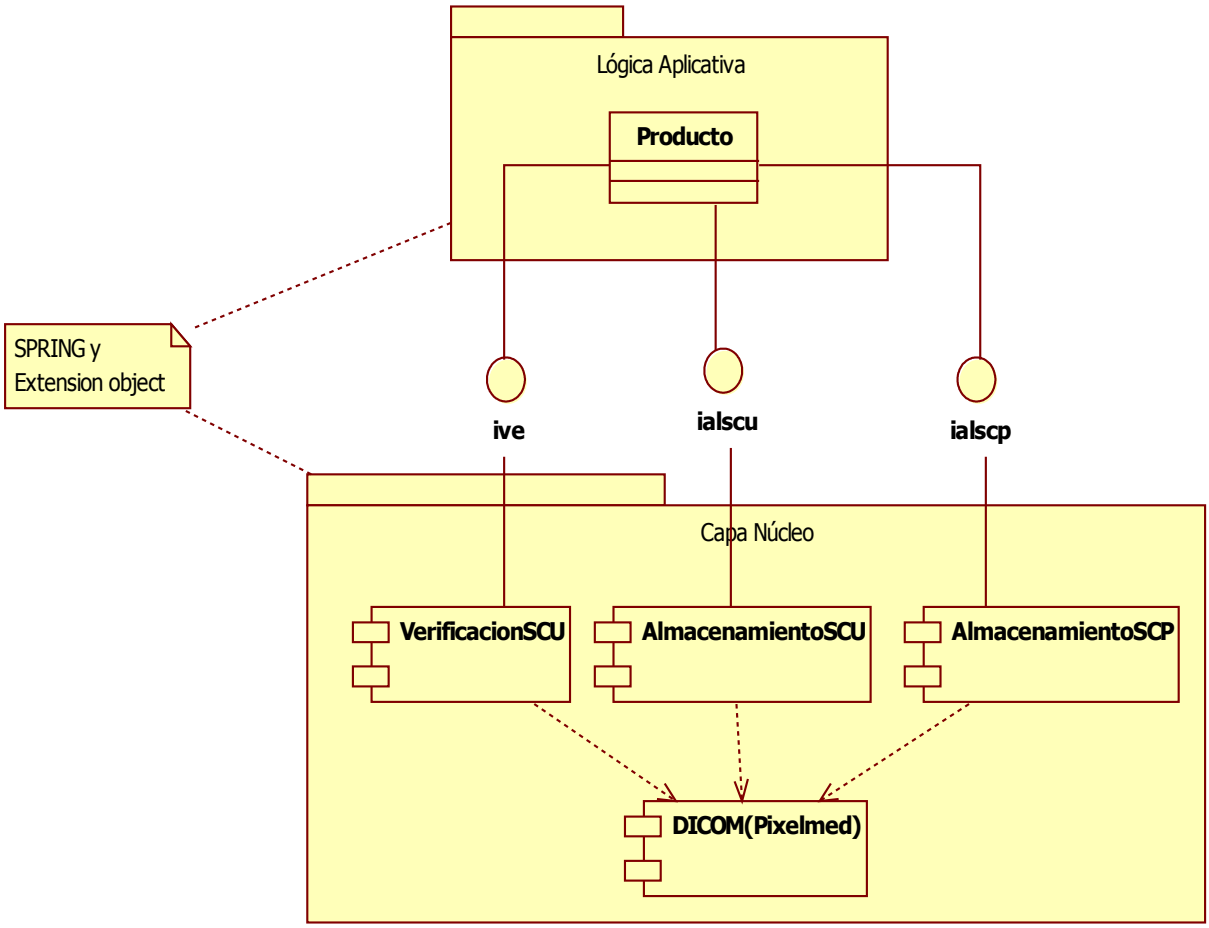

Figura 30: Modelo de capas para producto.

4.2.2 Interfaces de servicio DICOM.

Cada uno de los componentes presentados en la sección anterior implementa una interfaz particular que abstrae un servicio DICOM en sí, tanto en la modalidad cliente (SCU) como en la modalidad servidor (SCP). En la figura 30, las interfaces implementadas son ive= Interfaz verificación, Ialscu=Interfaz almacenamiento Service Class User, IalSCP=Interfaz almacenamiento Service Class Provider.

Esta elección de diseño tiene varias ventajas. Por un lado, permite desacoplar completamente la lógica de las entidades de aplicación de los detalles específicos de una implementación particular de DICOM. Por otro lado, permite sustituir fácilmente la implementación de DICOM.

Finalmente, permite que una entidad de aplicación juegue únicamente el papel de cliente

o de proveedor de servicio.

4.2.3 Configuración de los servicios del núcleo.

Como se mencionó en la sección 2.9, los distintos productos dentro de un PACS son llamados entidades de aplicación y cada una de ellas tiene asociado un *conformance* que especifica los servicios DICOM que se deben proporcionar y que requiere la entidad de aplicación. Con el fin de soportar la variabilidad de los productos, uno de los requerimientos principales del núcleo de la línea de producto es que éste permita configurar los servicios asociados al *conformance* de un producto específico de manera simple, es decir, sin necesitar de una recompilación del núcleo (atributo de calidad modificabilidad).

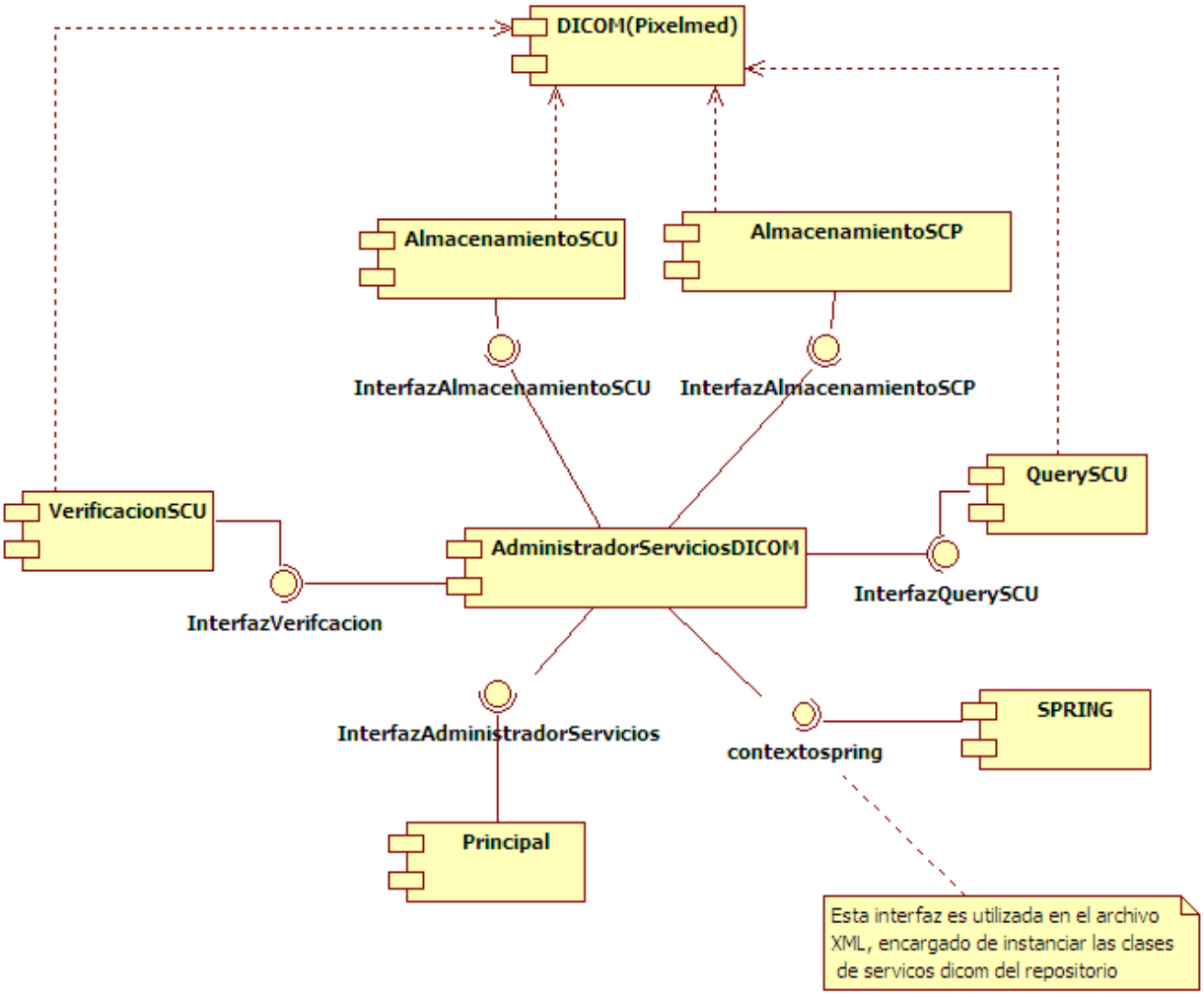

Figura 31: Diagrama de componentes para integrar servicios DICOM.

Para satisfacer dicha necesidad, en este trabajo se propone el uso de un *conformance* configurable que permite describir la lista de servicios DICOM asociados al producto, en un archivo en formato textual y fácilmente modificable por los desarrolladores. Este archivo es interpretado automáticamente para realizar una composición de los servicios requeridos por una entidad de aplicación al momento de su arranque. Este archivo puede ser visto como una versión automatizada del *conformance* textual que debe acompañar a toda entidad de aplicación DICOM. Como se explicó en la estrategia de producción (Sección 3.1.1.3), uno de los artefactos que el desarrollador debe crear para cada producto es el *conformance* configurable en formato XML, con lo que se estará cumpliendo con la descripción de cada producto que se desee generar a partir del núcleo arquitectural.

La interpretación del *conformance* configurable es realizada por un componente adicional del núcleo llamado AdministradorServiciosDicom (ver figura 31). Este componente se encarga de interpretar la especificación del *conformance* y de integrar los distintos componentes que implementan los servicios DICOM específicos a la entidad de aplicación. Una vez que se ensamblan los servicios de una entidad de aplicación, su núcleo puede ser visto como un componente de mayor granularidad que provee un número variable de interfaces. Este componente es utilizado por la lógica de negocios de la entidad de aplicación que debe descubrir dinámicamente qué servicios provee el núcleo (ya que es ensamblado al momento de arrancar la aplicación). El soporte a la navegación entre estas interfaces se realiza mediante la implementación del patrón *Extensión Objects* [GammaEO1995] que permite además soportar la evolución de las interfaces de servicio DICOM. La figura 32, ilustra el concepto de patrón *Extension Objects*. En dicha figura, cada una de las interfaces de extensión representa una interfaz de servicio DICOM. La implementación de componente se refiere al resultado de la composición de los componentes de granularidad fina a partir del *conformance* configurable.

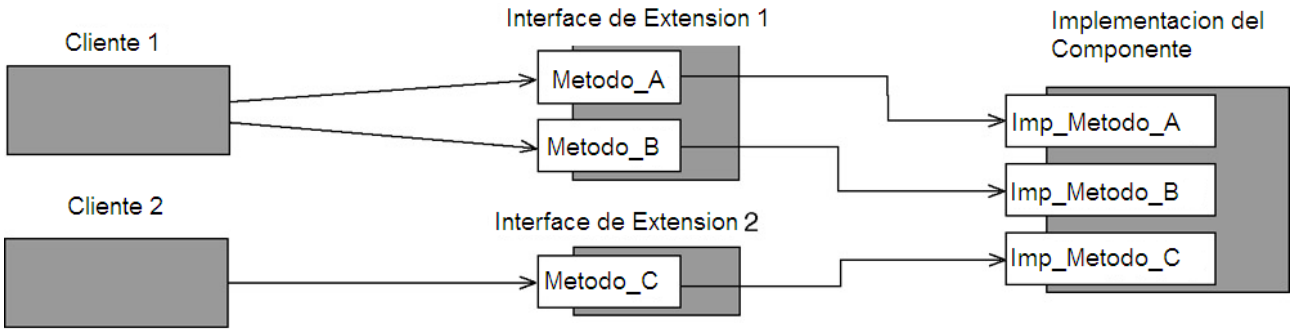

Figura 32: Patrón Extension Objects.

4.2.4 Repositorio de componentes.

Con el fin de soportar la integración de distintas configuraciones de núcleos de entidades de aplicación, es necesario que exista un repositorio que contenga tanto las distintas interfaces de los posibles servicios DICOM como los componentes que los implementan, como se muestra en la figura 33, para una entidad de aplicación cliente la inclusión de nuevos elementos en este repositorio y el enfoque de *conformance* configurable descrito anteriormente permiten soportar de manera sencilla la integración de nuevos servicios DICOM que estén disponibles para una nueva entidad de aplicación.

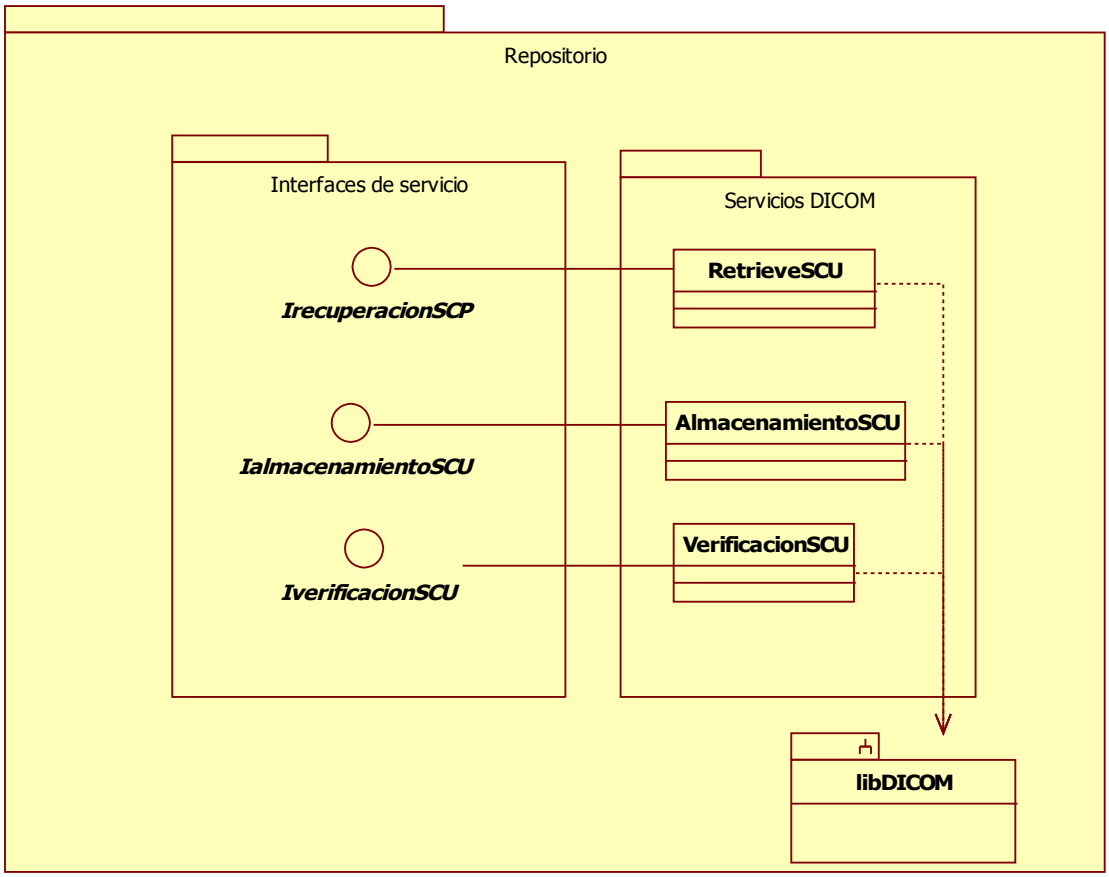

Figura 33: Repositorio de componentes de software para la arquitectura de línea de producto.

4.3 Implementación del núcleo.

A continuación se presentan los detalles de implementación del núcleo para la línea de producto.

4.3.1 Implementación del núcleo (Interfaces de los componentes del núcleo).

Como se describió en la sección relativa al diseño, desde un punto de vista conceptual, cada componente implementa una interfaz que representa un servicio DICOM en particular. A nivel de la implementación, estas interfaces se mapean directamente hacia interfaces Java.

Cada interfaz debe definir cuando menos los siguientes métodos:

- Método para soportar el servicio: este método se debe codificar en la clase que

implemente la interfaz; su objetivo es abstraer el servicio que proporciona la librería DICOM.

- Método para cambiar o agregar datos de configuración: este método sirve para configurar, en tiempo de ejecución, los parámetros que se requieran en cada servicio.
- Método para obtener el valor de las propiedades de configuración: con este método se obtendrá el estado de los valores de configuración para el servicio.

Como ejemplo se observa en la figura una interfaz para el servicio DICOM ECHO.

```
package interfazSPRING;
import java.util.Properties;
/**
 * Esta interfaz contiene los métodos para un
 * servicio de verificación DICOM para el núcleo arquitectrual PACS
 * @author Marco Antonio Núñez Gaona
 * @version v1.1
 *
 */
public interface IVerificacion {
/*
  * Este método debe invocar al método responsable de ejecutar el servicio en
  * librería DICOM que se utilice en la arquitectura.
  * @return true si la entidad proveedora de servicio respondió
  */
public boolean echoSCU();

 /*
  * Este metodo modifica las propiedades necesarias para soportar el  servicio
DICOM ECHO SCU.
  * claves       Descripcion
  * ------       --------------
  * "host"        Dirección  IP  de  la  entidad  de  aplicación  proveedor  de
servicio
  * "port"        Puerto de la entidad de aplicación proveedor de servicio
  * "calledAET"   Nombre  entidad  de  aplicación  proveedora  del  servicio  DICOM
longitud maxima 16 caracteres (remota)
  * "callingAET"  Nombre entidad de aplicación que utiliza el servicio (
  */
 public void modificaPropiedad(String clave, String valor);

 /*
  * Este metodo regresa los datos actuales de configuracion
 */
 public Properties regresaDatosConfiguracion();

}
```

Figura 34: Interfaz para el servicio DICOM ECHO.

4.3.2 Implementación de los componentes.

La implementación de los componentes se realiza a través de una clase Java que a su vez implementa alguna de las interfaces de servicios DICOM.

Dentro de esta clase se llama directamente a la implementación de la librería DICOM y de esta forma se aísla la lógica de negocio de la entidad de aplicación, del uso de una implementación particular del estándar DICOM.

Como se observa en la figura 34, es importante implementar la interfaz que contiene los métodos para configurar y hacer la llamada al servicio DICOM que se encuentra implementado en las librerías. Los puntos importantes a destacar es el uso de las interfaces del repositorio, la llamada al servicio que provee la librería DICOM utilizada y la configuración requerida por éste último.

El desarrollador solo debe saber como realizar la llamada del servicio DICOM que se encuentra codificado en las librerías de comunicación.

A continuación se muestra un ejemplo de implementación del servicio DICOM ECHO.

```
package serviciosDICOM;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

//SE HACE USO DE LA INTERFAZ QUE SE ENCUENTRA EN EL REPOSITORIO
import interfazSPRING.IVerificacion;
import com.pixelmed.network.VerificationSOPClassSCU;

public class VerificacionSCU implements IVerificacion{

   Properties propiedadesconfiguracion=null;
   private boolean seguridad=false;
   private int niveldepuracion=0;

//CONSTRUCTOR CARGA PROPIEDADES DE CONFIGURACION
   public VerificacionSCU() {
      try{
         propiedadesconfiguracion=cargaPropiedades(AlmacenamientoSCU.class.getR
esourceAsStream("res/Verificacion.properties"));
      }catch(Exception ex){
         ex.printStackTrace();
      }
   }

   public Properties regresaDatosConfiguracion(){
      return propiedadesconfiguracion;
   }

   private static Properties cargaPropiedades(InputStream inputstream) throws
IOException{
      Properties propiedades = new Properties();
      try{
         propiedades.load(inputstream);
      }finally{
         inputstream.close();
      }
      return propiedades;
   }

   /**
    * CODIFICACION DEL METODO QUE UTILIZA EL SERVICIO DICOM
    * IMPLEMENTADO EN LAS LIBRERIAS DICOM
    * @param      host...........: dirección ip proveedor de servicio
    * @param      puerto.........: numero de puerto de proveedor de servicio
    * @param      CallingAET.....: Titulo de entidad de aplicación cliente
    * @param      CalledAET......: Titulo de entidad de aplicación servidor
    * @param Seguridad......: Asegura comunicación segura
    * Llamada: VerificaConexiconSCU();
    **/
   public boolean echoSCU() {
      try {
      /* Se hace la llamada el servicio DICOM con los parámetros necesarios
        de la entidad de aplicación que proporciona el servicio
       (Verificacion-SCP), que se encuentra configurada para este servicio
      (Verificacion-SCU). En este caso pixelmed implementa la clase de
      verificación de la siguiente forma:
```

```
         public VerificationSOPClassSCU(String hostname,int port,
         String calledAETitle,String callingAETitle,boolean secureTransport,
         int debugLevel)
      */

      new VerificationSOPClassSCU(propiedadesconfiguracion.getProperty("host"),
         Integer.parseInt(propiedadesconfiguracion.getProperty("port")),
         propiedadesconfiguracion.getProperty("calledAET"),
         propiedadesconfiguracion.getProperty("callingAET"),seguridad,niveldepu
racion);
      }catch (Exception e) {
         e.printStackTrace(System.err);
         return false;
      }
      return true;
   }
}
```

Figura 35: Implementación servicio DICOM ECHO.

4.3.3 Integración de los componentes a partir del *conformance* configurable.

Como se mencionó previamente, la integración de los componentes del núcleo se hace a partir de la información de *conformance* que es traducida hacia un *conformance configurable*. Este *conformance* configurable es utilizado por el componente del núcleo llamado AdministradorServiciosDicom para ensamblar el núcleo de un producto específico. A nivel de la implementación, esto se hace mediante el uso del framework *Spring,* el cual realiza la instanciación y conexión de los componentes a partir de la representación en XML del *conformance*, llamado *ApplicationContext*. *Spring* ofrece un mecanismo de inversión de control e inyección de dependencias, que tiene como particularidad que los componentes que son ensamblados, no toman la iniciativa de realizar las conexiones, sino que estas son “inyectadas” por *Spring* a través de métodos “*setter”* provistos por los componentes, que permite que sean conectados en distintas configuraciones. El uso de *Spring* permite crear un elemento ”ensamblador de núcleos” sin la necesidad de recompilar el producto. Por otro lado, el archivo usado por *Spring* permite representar el c*onformance* configurable descrito anteriormente.

La estructura del archivo de c*onformance* configurable es la siguiente ver figura 36.

Para cada servicio DICOM que la entidad de aplicación requiera, es necesario crear una instancia del componente que implementa dicho servicio, también conocido como *bean*.

Dicha instancia tiene las siguientes características:

1. El nombre de la clase del componente que implementa un servicio particular por ejemplo: *class*=”serviciosDICOM.nombreservicio”
2. Un identificador único por ejemplo: *id*=”nombre”
3. Parámetros de configuración de acuerdo al servicio DICOM, por ejemplo: *property name*=”nombre parámetro” *value*=”valor parámetro”
4. El nombre de las entidades de aplicación involucradas (máximo 16 caracteres).
5. El número de puerto para comunicación para cada entidad de aplicación.
6. Las direcciones IP de los nodos participantes.

En la figura 36 se muestra un breve ejemplo de la semántica y sintaxis del archivo.

Este ejemplo muestra un ***conformance* configurable** que utiliza los servicios DICOM ECHO y DICOM STORAGE en su modalidad de usuarios de servicio.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
      <bean id="echoscu" class="serviciosDICOM.VerificacionSCU">
            <property name="port" value="104"/>
            <property name="called" value="ECHO-SCP"/>
            <property name="host" value="192.168.10.101"/>
      </bean>

      <bean id="storagescu" class="serviciosDICOM.AlmacenamientoSCU">
            <property name="port" value="104"/>
            ................................
            ................................
      </bean>

      ........
      ........

</beans>
```

Figura 36: Ejemplo de conformance configurable.

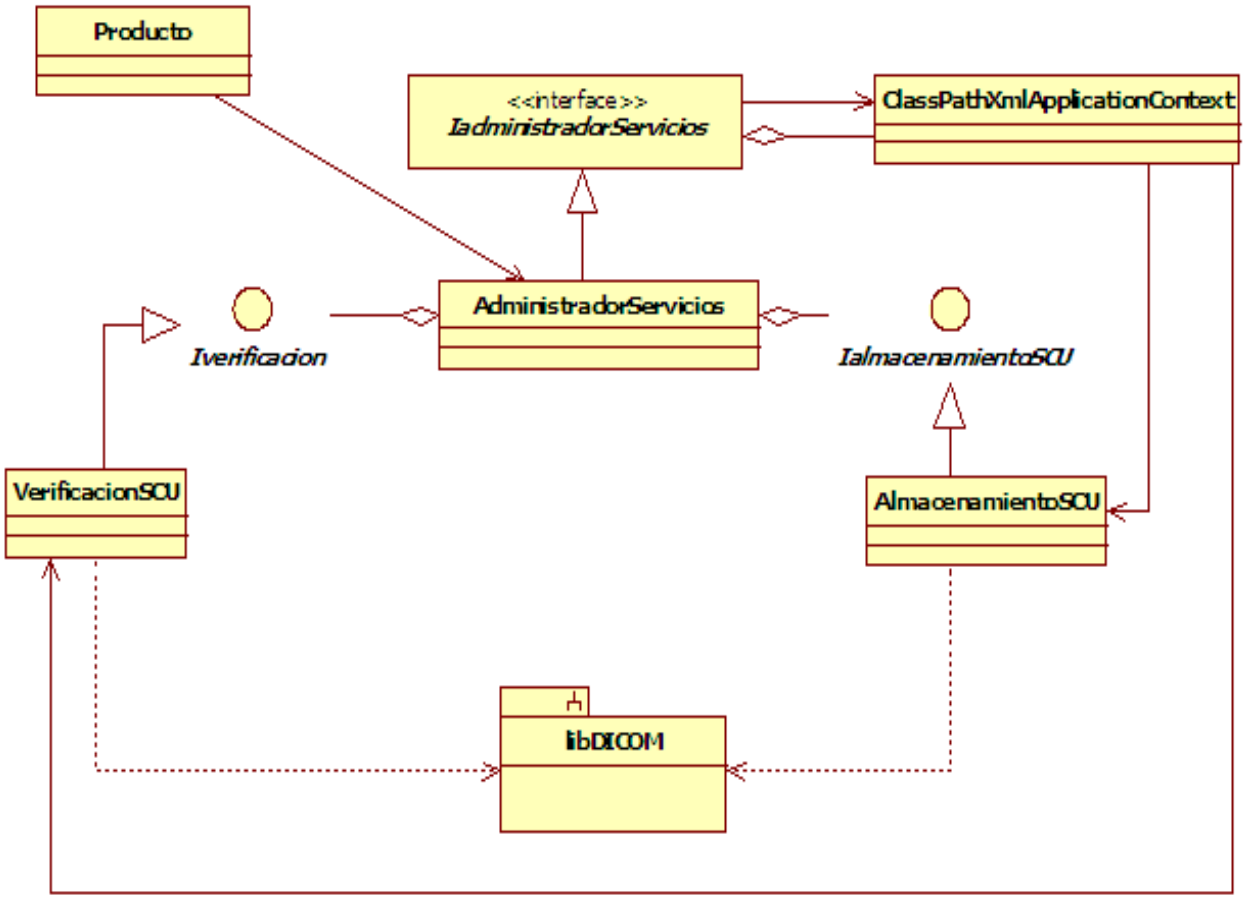

Figura 37: Agregación de servicios DICOM basado en el patrón de extensión de objetos.

La figura 37 muestra como un producto registra un conjunto de servicios DICOM; implementando las interfaces necesarias se hace uso de las librerías de comunicación DICOM y la inyección de dependencias a través de *Spring* por medio del *conformance* configurable, el cual es interpretado por la clase *ClassPathXmlApplicationContext.*

Con el fin de facilitar la configuración de los diversos servicios DICOM, en el repositorio se almacenan archivos que contienen las propiedades de los componentes y que permiten realizar dicha configuración.

Un ejemplo de este tipo de archivos es la figura 38.

```
#Propiedades de  VerificacionSCU
port=104
callingAET=ECHO_SCU
calledAET=ECHO_SCP
host=192.168.10.184
```

Figura 38: Archivo de configuración para un servicio DICOM ECHO.

En la figura 38 se muestra que un proveedor de servicio tiene la dirección IP

192.168.10.184 se encuentra activo por el puerto 104 y se llama ECHO_SCP. El usuario de servicio se llama ECHO_SCU.

4.3.4 Extensión del núcleo.

Los distintos componentes que implementan servicios DICOM se almacenan en el repositorio, descrito previamente, que se materializa como un directorio que contiene archivos con código binario.

En la figura 39, se presentan algunos ejemplos de interfaces y clases correspondientes a diversos servicios DICOM disponibles en el repositorio.

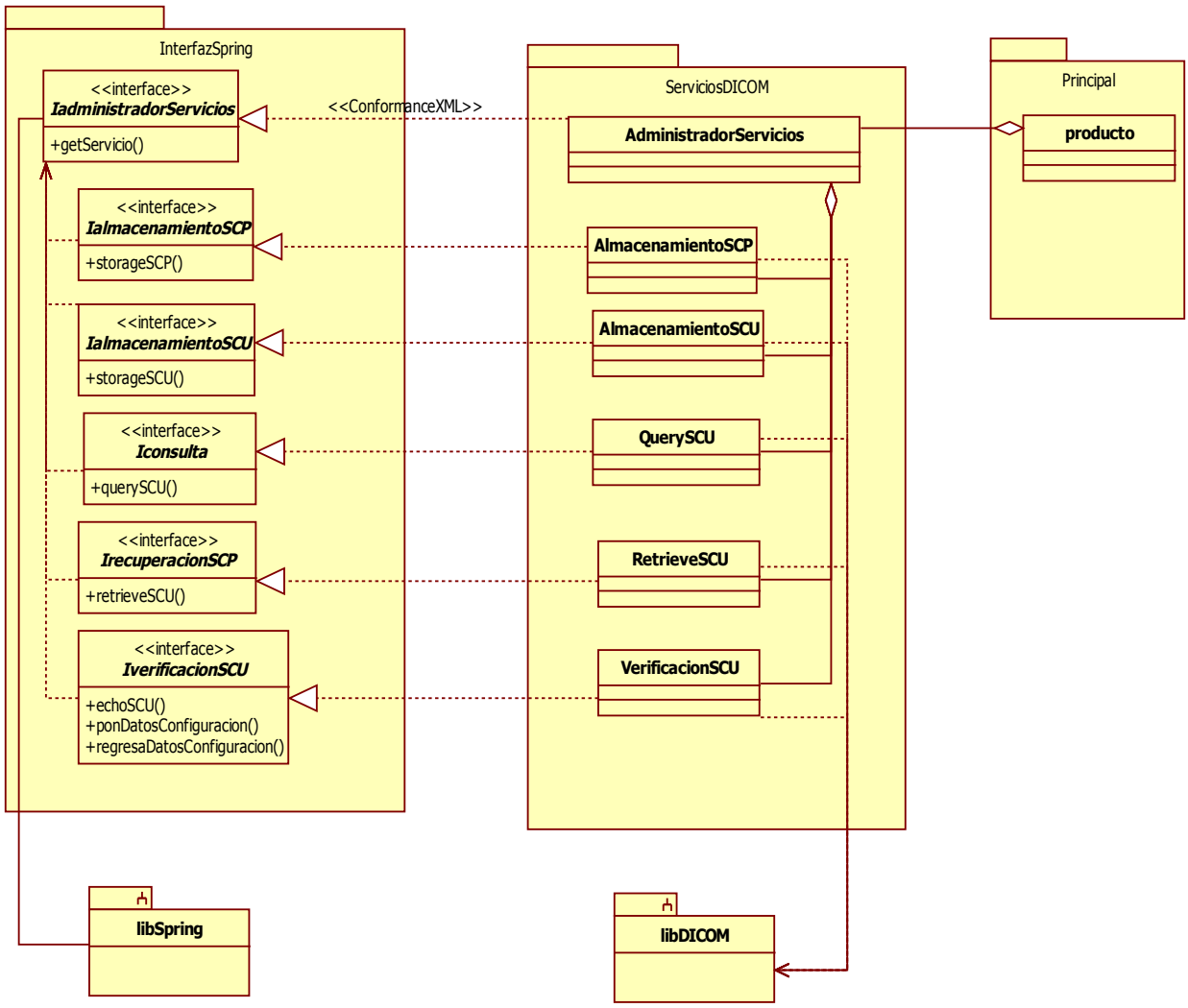

Figura 39: Modelo de clases del repositorio de la línea de producto.

A continuación se detallan los distintos elementos correspondientes a los componentes

que implementan los servicios DICOM.

4.3.5 Administrador de servicios.

El repositorio es el contenedor principal de todos los elementos o componentes que se utilizan para la generación de productos. Este se divide en interfaces y servicios. En la parte de interfaces existe la interfaz *IAdministradorServicios* cuya implementación la realiza el administrador de servicios utilizando el patrón *Extension Objects*. Aquí mismo se debe definir una interfaz por cada servicio DICOM que se desee agregar en una entidad de aplicación por medio del *conformance* configurable.

El esquema anterior facilita la implementación del patrón de *Extension Objects* así como la configuración de los productos, de esta forma se logra que la arquitectura sea independiente de cualquier implementación DICOM.

En las figuras 40 y 41 se presenta la implementación del *AdministradorServicios* y también se presenta el código correspondiente al patrón *Extension Objects* con el fin de aclarar como se debe hacer uso en cualquier entidad de aplicación.

```
package serviciosDICOM;
import interfazSPRING.IAdministradorServicios;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/*
 * Esta clase se implementa como un patron extesion Object (GAMMA)
 * Se implementan varios objetos sin modificar la interfaz.
 * el cliente hace un cast dinamico al obtener el servicios deseado
 * cuyo identificador se toma directamente del archivo xml (Conformance de la
aplicacion)
 */
public class AdministradorServicios implements IAdministradorServicios{
 private ClassPathXmlApplicationContext contexto;
 private String[] nombredeservicios=null;

public AdministradorServicios(String nombreconformanceconfigurable){
 try{
  System.out.println("Cargando interfaz Spring");
 contexto = new ClassPathXmlApplicationContext(new String[]
{nombreconformanceconfigurable});
  System.out.println("Se cargo interfaz spring adecuadamente");
 }catch(Exception e){
   e.printStackTrace();
 }
  nombredeservicios=contexto.getBeanDefinitionNames();
 }

 public Object getServicio(String id){
  return contexto.getBean(id);
 }

 public String[] getServiciosRegistrados(){
  return nombredeservicios;
 }
}
```

Figura 40: Código de implementación en Java del componente AdministradorServicios.

Esta parte de código muestra como hacer uso del administrador de servicios así como la recuperación del servicio deseado para su invocación. Se observa también como la clase *ClassPathXmlApplicationContext* es responsable de cargar el *conformace* configurable.

```
AdministradorServiciosadmonserviciosdicom = new
AdministradorServicios("serviciosDICOMspring.xml");
/* Dependiendo de la definición de servicios en el conformance digital ejemplo:
* echoscu, storagescu, storagescp, queryscu.
*
* La referencia de servicio se solicita al administrador de
* servicios por medio de una cadena que se define en el conformance digital
* como el identificador de servicio (bean id=”nombre”)
*/
  Object serviceRef=serviciosdicomsoportados.getServicio(cadenaservicio);
  if(serviceRef != null) {
     if(serviceRef instanceof IVerificacion){
      IVerificacion verificacion = (IVerificacion)serviceRef;
      boolean bandera=verificacion.echoSCU();
      Properties tmp=verificacion.regresaDatosConfiguracion();
      String echoscp=tmp.getProperty("calledAET");

     }else if(serviceRef instanceof IAlmacenamientoSCU){
       IAlmacenamientoSCU almacenamientoarchivo=(IAlmacenamientoSCU)serviceRef;
       almacenamientoarchivo.modificaPropiedad("callingAET", "almscu");
       boolean bandera=almacenamientoarchivo.storageSCU(dicomFileName);
       Properties tmp=almacenamientoarchivo.regresaDatosConfiguracion();
       String aescp=tmp.getProperty("calledAET");

    }else if(serviceRef instanceof IAlmacenamientoSCP){
      IAlmacenamientoSCP servidoralmacenamiento=(IAlmacenamientoSCP)serviceRef;
      boolean bandera=servidoralmacenamiento.storageSCP();
      Properties tmp=servidoralmacenamiento.regresaDatosConfiguracion();
      String aescp=tmp.getProperty("calledAET");
      int puerto=Integer.parseInt(tmp.getProperty("port"));

     }else if(serviceRef instanceof IConsulta){
      IConsulta consultascu=(IConsulta)serviceRef;
      try{
       listapacientes=consultascu.consultaPacienteEstudio(listapacientes);
      }catch(Exception ex){
       ex.printStackTrace();
      }

      if(listapacientes!=null){
       Tablas listapac= new Tablas(listapacientes,consultascu);
       listapac.setLocation(200, 150);
       listapac.setVisible(true);
      }else{
      dialogoResultado(false,"No es posible ejecutar la consulta ");
      }
     }
   }
  }
 }
```

Figura 41: Código de implementación en Java del patrón Extension Objects.

En el fragmento de código anterior lo importante a destacar es lo siguiente:

1. Si se desea recuperar un servicio, solo hay que pedir al administrador de servicios el nombre del servicio deseado: bean Id=”nombre” definido en el *conformance* configurable.
2. La referencia del servicio se compara con las interfaces que se encuentran en el repositorio general. Si esta referencia pertenece al repositorio se hace un *casting* de clase y se ejecuta el servicio llamando al método correspondiente [GammaEO1995].
3. También es importante destacar que es posible cambiar los atributos de configuración para cada servicio lo que facilita la administración del producto final.

4.4 Escenario de construcción de un producto particular.

El conjunto de actividades que se plantean en el desarrollo del núcleo de la línea de producto se pueden enlazar con el proceso unificado de la siguiente forma.

Las líneas de producto proponen la creación de un alcance (*scoping*), es decir se definen los productos que se generan a partir de esta y cuáles quedan fuera, o sea dimensionar el tipo de producto para PACS. Esto se puede realizar en la etapa de concepción con RUP en donde se define una visión del proyecto, la cual consiste en acotar las fronteras del sistema y analizar los elementos del producto que se desee desarrollar.

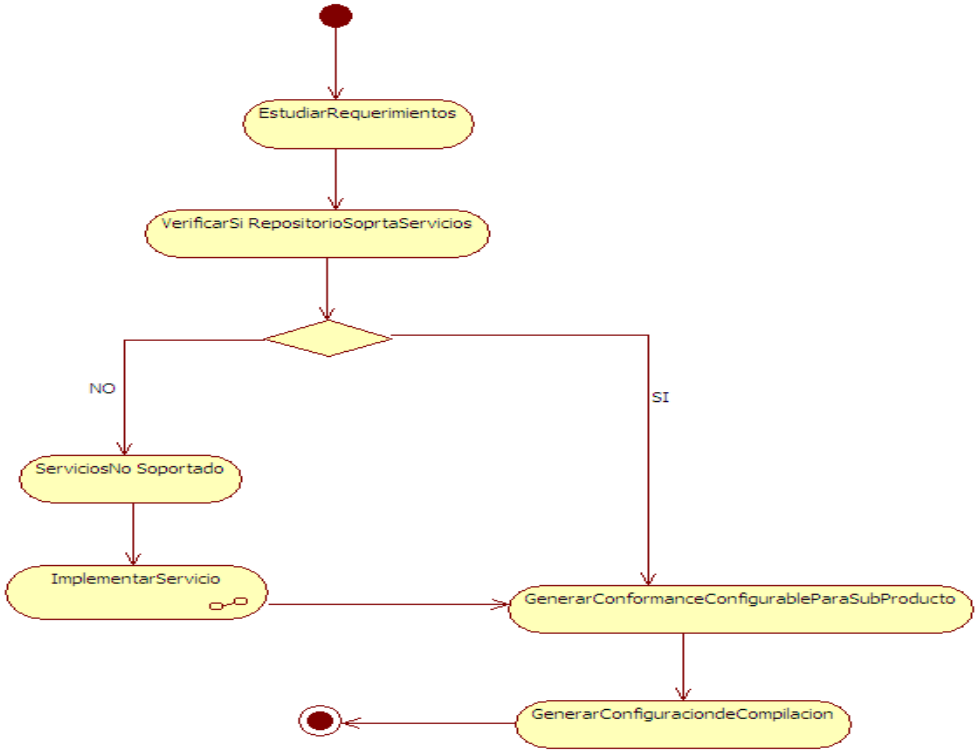

Figura 42: Diagrama de secuencias para generar los productos de un PACS.

Los puntos de variación para cada producto se deben considerar en la definición del *conformance* configurable, reflejando el conjunto de servicios DICOM que cada producto (entidad de aplicación) deberá soportar. Para poder generar algún producto, es necesario que el desarrollador realice un conjunto de actividades. En la figura 42 se presenta el diagrama general de actividades relacionado con la etapa de elaboración y construcción definido en la sección 3.1.1.3 y la Figura 22, cuyo objetivo es mostrar el trabajo que el desarrollador debe realizar para construir cada producto particular. A continuación se hace una descripción de los puntos importantes.

Actividad 1 EstudiarRequerimientos: en esta actividad el desarrollador debe estudiar los requerimientos que el arquitecto ya definió para cada producto, con el fin de generar el *conformance* configurable con las especificaciones de todos los servicios DICOM que debe soportar la entidad de aplicación (descripción del producto).

Actividad 2 VerificaSiRepositorioSoportaServicios: el desarrollador debe analizar si las librerías de comunicación soportan los servicios requeridos, analizar si el repositorio

cuenta con las interfaces para cada servicio y analizar si se implementó el servicio.

Actividad 3 GenerarConformanceConfigurableParaproducto: si la verificación de servicios fue satisfactoria, la integración de servicios consiste en generar el archivo de *conformance* configurable con las especificaciones que ya se mencionaron en el punto 4.3.3. Es importante destacar que por medio de esta actividad se cumplen con los puntos de variación que se define en la especificación de línea de producto para implementar varios productos a partir de un mismo núcleo (variabilidad de acuerdo al tipo de sistema deseado).

Actividad 4 GenerarConfiguraciondeCompilacion: esta actividad es para generar un archivo de compilación *ant* con las librerías necesarias para cada subproducto, dando como resultado el conjunto de aplicaciones que pasan a la etapa de pruebas y transición.

4.5 Síntesis del capítulo.

En este capítulo se presentaron los detalles relativos al diseño e implementación del núcleo para la línea de producto para PACS. Los puntos importantes a destacar son:

- Un núcleo para una entidad de aplicación específica, se ensambla a partir de la composición de componentes binarios que implementan interfaces y que representan servicios DICOM individuales tanto en modalidad de provisión del servicio, como de usuario del servicio. Dichas interfaces permiten aislar la lógica de negocios de una entidad de aplicación específica de la implementación particular del estándar DICOM.
- El núcleo es ensamblado a partir de un documento textual o *conformance* configurable, que es escrito por el desarrollador de una entidad de aplicación. Dicho documento corresponde al *conformance* de la entidad de aplicación y describe, en formato XML, cuáles son los servicios asociados a la entidad, así como su configuración.
- En tiempo de ejecución el *conformance* configurable es interpretado por un componente que se basa en *Spring* y que realiza la integración de los componentes del núcleo, y de la entidad de aplicación.
- El conjunto de componentes e interfaces, que representan los elementos o bienes (*core assets)* de la línea de productos, se almacenan en un repositorio y su número

no está limitado, por lo cual es posible soportar la introducción futura de nuevos servicios.

- La entidad de aplicación interactúa con el núcleo, y puede navegar a través de sus interfaces mediante la implementación del patrón *Extension Objects*
- Los PACS usan tipos distintos de entidades de aplicación, correspondientes a sus necesidades. El identificar los servicios DICOM específicos a las entidades de aplicación, le permite al desarrollador de una entidad de aplicación poder armar de forma sencilla un núcleo que soporte su entidad de aplicación.
- La construcción de una entidad de aplicación usando el enfoque de línea de producto propuesto en este trabajo se puede realizar de forma sistemática siguiendo el proceso descrito.

El capítulo siguiente presenta una evaluación de la propuesta a través del desarrollo de un ejemplo de entidad de aplicación.

5 Prototipo de validación.

En este capítulo se presenta un ejemplo de ejecución del proceso para generación de un producto de acuerdo al proceso descrito anteriormente. En este ejemplo, se detallan las distintas actividades que un desarrollador debe realizar y artefactos que se deben obtener para generar un producto de visualización de imágenes médicas que puede formar parte de un PACS. A través de un prototipo sencillo se plantea la integración de servicios DICOM, tratando de hacer más comprensible el uso de la arquitectura de línea de producto PACS. Se muestra como hacer uso de los elementos del núcleo utilizando el repositorio central para implementar un conjunto de servicios DICOM de interés para un producto de visualización.

5.1 Construcción de un visualizador DICOM.

En esta sección se presenta una instancia de la estrategia de producción y el plan de producción, propuestos en el capítulo 3.

5.1.1 Requerimientos del prototipo de visualización (Producto Visualización).

Se requiere construir un visualizador de imágenes DICOM que permita desplegar en pantalla imágenes médicas provenientes de las siguientes modalidades: CT, MR, NM, CR, US.

El prototipo de visualización debe presentar al usuario un diálogo por medio del cual se seleccionen las imágenes de un paciente e inmediatamente presentar la imagen seleccionada en un panel.

El sistema debe soportar los siguientes servicios DICOM.

- ECHO_SCU.
- STORAGE_SCU.
- STORAGE_SCP.
- QUERY_SCU.

Así mismo, cada servicio debe soportar la configuración de parámetros para comunicarse

con otra entidad de aplicación. El visualizador debe permitir al usuario seleccionar cualquier servicio por medio de un menú de opciones.

Los servicios antes listados son indispensables en un sistema de visualización debido a que el médico radiológico debe recibir o consultar imágenes provenientes de diferentes modalidades o sistema de almacenamiento. Por otra parte, también puede enviar la interpretación de imágenes significativas que apoyan el diagnóstico clínico.

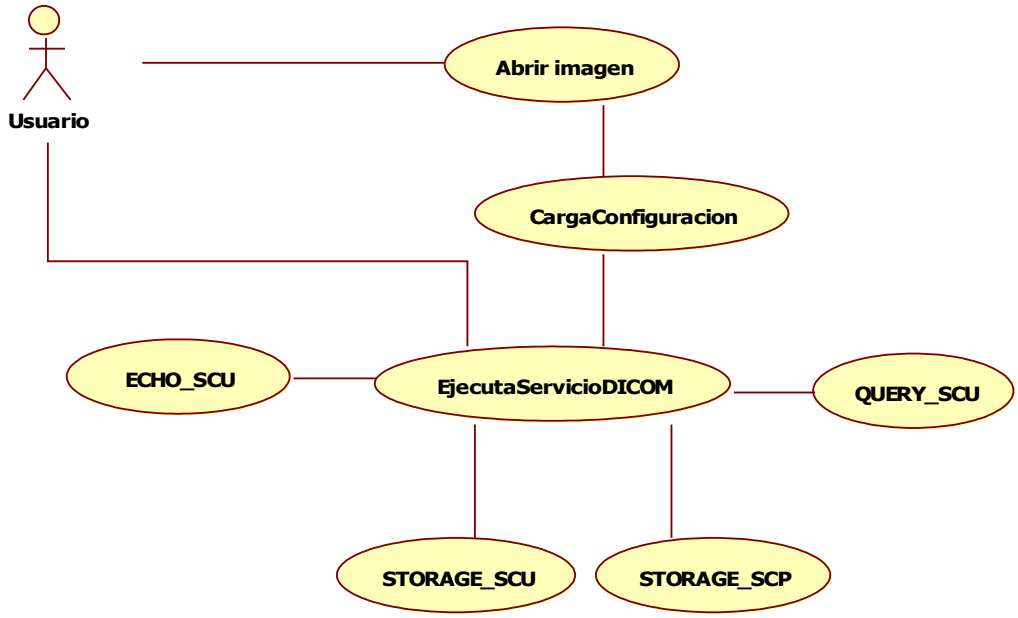

Figura 43: Diagrama de casos de uso para prototipo de visualización.

La figura 43 presenta el diagrama de casos de uso para el prototipo de visualización, como podemos observar el prototipo contempla un caso de uso responsable de ejecutar los servicios DICOM que soportará la aplicación. Este diagrama debe ser escrito por el desarrollador, ya que le facilitará realizar la descripción de cada producto y se comprenderá la funcionalidad total.

Es responsabilidad del desarrollador implementar las interfaces necesarias para los servicios DICOM del visualizador. Si las interfaces ya se implementaron con anterioridad solo es necesario hacer una copia desde el repositorio.

5.1.2 Generación del *conformance* configurable.

Es importante no perder de vista que el objetivo es implementar un producto a partir de la arquitectura de línea de producto, para la cual es necesario crear un *conformance* que

describe el comportamiento del visualizador. Este documento servirá como plantilla para identificar la funcionalidad, en conjunto con los casos de uso de un visualizador. A su vez, el desarrollador lo utilizará como referencia para conocer los servicios que se integrarán en el producto a implementar.

La figura 44 presenta un diagrama de un *conformance* para un visualizador de imágenes, este diagrama representa también la descripción de un producto de visualización que debe generarse antes de construir el producto.

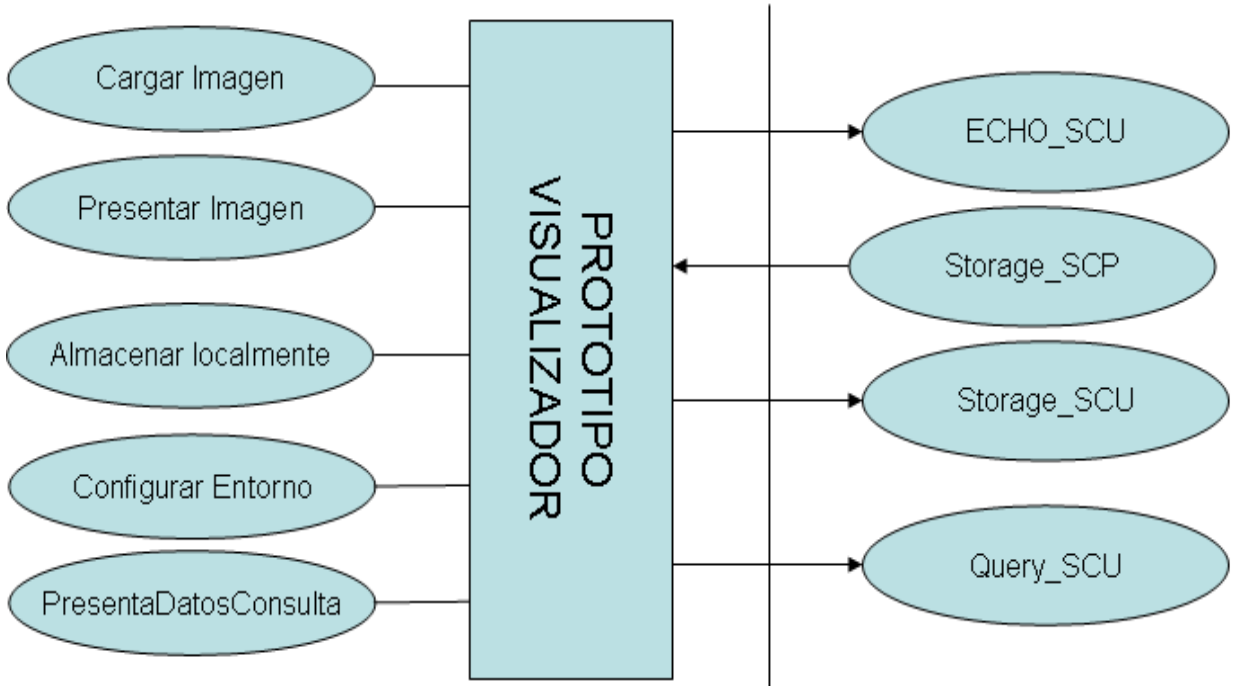

Figura 44: Diagrama conformance prototipo visualizador.

Utilizando la descripción del *conformance* para el producto de visualización, se debe especificar el *conformance* configurable, en formato XML, este archivo debe nombrarse como: “ServiciosDICOMVisualizador.XML”. El contenido se muestra en la figura 45. El archivo contiene 4 descriptores de clases de servicio, donde cada una de ellas representa la funcionalidad de un caso de uso para cada servicio DICOM soportado, también se considera una configuración por omisión para cada uno de estos servicios.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-
beans.dtd">

	<!-- Se indica cuales son los servicios DICOM que se utilizaran del repositorio -->
<beans>
	<bean id="echoscu" class="serviciosDICOM.VerificacionSCU">
		<property name="port" value="104"/>
		<property name="callingAET" value="ECHO_SCU"/>
		<property name="calledAET" value="STORAGE_SCP"/>
		<property name="host" value="localhost"/>
	</bean>

	<bean id="storagescu" class="serviciosDICOM.AlmacenamientoSCU">
		<property name="port" value="104"/>
		<property name="callingAET" value="STORAGE_SCU"/>
		<property name="calledAET" value="STORAGE_SCP"/>
		<property name="host" value="localhost"/>
	</bean>

	<bean id="storagescp" class="serviciosDICOM.AlmacenamientoSCP">
		<property name="port" value="104"/>
		<property name="calledAET" value="STORAGE_SCP"/>
		<property name="rutaservidor" value="c:\\tmp"/>
	</bean>
	<bean id="queryscu" class="serviciosDICOM.Query">
		<property name="port" value="4006"/>
		<property name="calledAET" value="lx-mr"/>
		<property name="callingAET" value="QUERY_SCU"/>
		<property name="host" value="localhost"/>
	</bean>

</beans>
```

Figura 45: Conformance configurable para prototipo de visualización.

5.1.3 Implementación de interfaces para servicios DICOM.

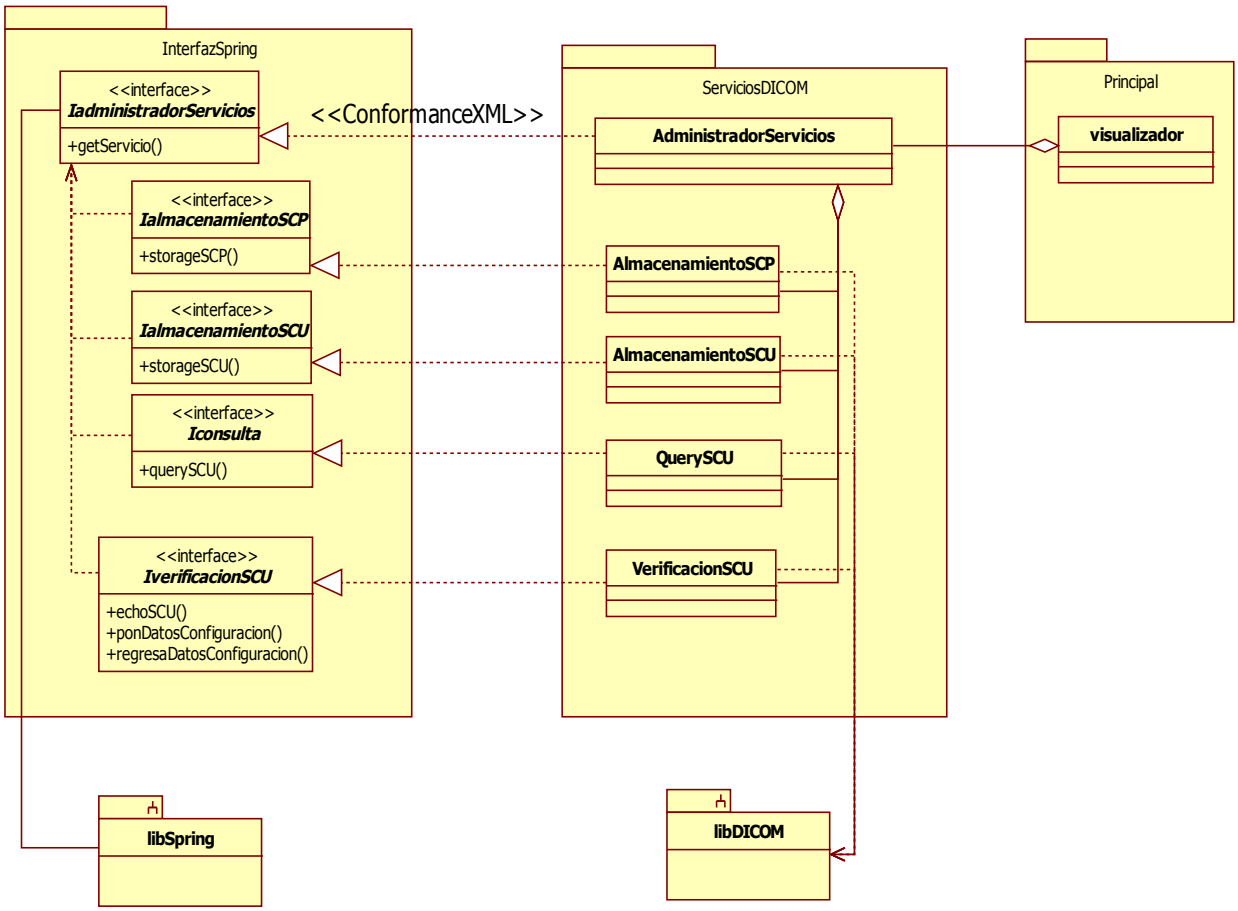

Figura 46: Recuperación del núcleo para el prototipo visualizador.

Para lograr la independencia de cualquier implementación DICOM, se debe generar un conjunto de interfaces para los diferentes servicios que los productos del PACS deberán soportar. Para el prototipo de visualización (ver figura 46) se deben implementar las siguientes interfaces:

IadministradorServicio.java: es responsable de proporcionar un método que funcione como una fabrica de servicios. Es decir, su tarea es proporcionar el servicio adecuado cuando se lo solicite una aplicación.

Iverificacion.java: es responsable de proporcionar métodos para soportar la implementación y configuración del servicio *DICOM ECHO-SCU*.

IalmacenamientoSCU.java: es responsable de proporcionar un método para soportar la implementación y configuración del servicio *DICOM STORAGE-SCU*.

IalmacenamientoSCP.java: es responsable de proporcionar un método para soportar la implementación y configuración del servicio *DICOM STORAGE-SCP*.

Iconsulta.java: es responsable de proporcionar un método para soportar la implementación y configuración del servicio *DICOM QUERY-SCU*.

En el repositorio de servicios DICOM deben existir las siguientes clases JAVA:

AdministradorServicios.java: implementa la interfaz *IadministradorServicio.java.* Su responsabilidad es hacer la llamada al método de las librerías *SPRING* para cargar el *conformance* configurable que se definió a través del archivo XML. Esta clase se implementa como un patrón E*xtension objects* con el fin de implementar varios objetos sin modificar la interfaz.

VerificacionSCU.java: implementa la interfaz *Ivericacion.java* para llamar al método que ejecuta el servicio *DICOM ECHO_SCU* en las librerías DICOM seleccionadas.

AlmacenamientoSCU.java: implementa la interfaz *IalmacenamientoSCU.java* para llamar al método que ejecuta el servicio *DICOM STORAGE_SCU* en las librerías DICOM seleccionadas.

AlmacenamientoSCP.java: implementa la interfaz *IalmacenamientoSCP.java* para llamar al método que ejecuta el servicio *DICOM STORAGE_SCP* en las librerías DICOM seleccionadas.

Query.java: implementa la interfaz *Iconsulta.java* para llamar al método que ejecuta el servicio *DICOM QUERY_SCP* en las librerías DICOM seleccionadas.

5.1.4 Uso de los servicios DICOM.

Para que un elemento del producto PACS contenga el conjunto de servicios DICOM que se definieron en el *conformance* configurable y la implementación de las diferentes interfaces, es necesario generar una instancia de la clase *AdministradorServicios* y posteriormente generar un método o clase que permita ejecutar el servicio DICOM deseado con base en una orden o decisión. El fragmento de código de la figura 47 ejemplifica cómo lograr la implementación de los servicios para el visualizador.

```
public class Visualizador {

public Visualizador() {
AdministradorServicios admonserviciosdicom = new
AdministradorServicios("serviciosDICOMspring.xml");
//Cadena de servicio puede tomar los valores    //
[”echoscu”,”storagescu”,"storagescp",”queryscu”]
 Object serviceRef=serviciosdicomsoportados.getServicio(cadenaservicio);
  if(serviceRef != null) {
   if(serviceRef instanceof IVerificacion){
      IVerificacion verificacion = (IVerificacion)serviceRef;
      boolean bandera=verificacion.echoSCU();
      Properties tmp=verificacion.regresaDatosConfiguracion();
      System.out.println(“Resultado”+bandera)
  }else if(serviceRef instanceof IAlmacenamientoSCU){
   ..............
   ..............
   ..............
  }else if(serviceRef instanceof IAlmacenamientoSCP){
   ..............
   ..............
   ..............
  }else if(serviceRef instanceof IConsulta){
   ..............
   ..............
   ..............
  }
  //Main method
  public static void main(String[] args) {
    new Visualizador();
  }
}
```

Figura 47: Código para ejecutar los servicios DICOM para el prototipo.

El objetivo del código es solicitar al *administrador de servicios* la clase de servicio responsable de proveer el servicio DICOM (if(serviceRef != null)). Si esta clase esta presente en el repositorio, se regresa su referencia y posteriormente hay que verificar cual es la clase para hacer un casting adecuado (if(serviceRef instanceof IAlmacenamientoSCU)) y llamar al servicio que corresponde :

IAlmacenamientoSCU almacenamientoarchivo=(IAlmacenamientoSCU)serviceRef;

boolean bandera=almacenamientoarchivo.storageSCU(dicomFileName);

5.1.5 Ensamblado del visualizador.

Para integrar el conjunto de archivos y librerías que se incorporan al prototipo es recomendable tener un entorno de desarrollo como puede ser eclipse [ECLIPSE2007] así como herramientas de compilación [ANT2007] y ejecución para el proyecto. En la figura

48 se observa en la columna izquierda los paquetes y las clases necesarios para cumplir con la funcionalidad del prototipo de visualización de imágenes médicas. El archivo antvisualizador.xml, es el responsable del ensamblado y generación del producto final. Cada desarrollador será responsable de generar este archivo con los parámetros necesarios.

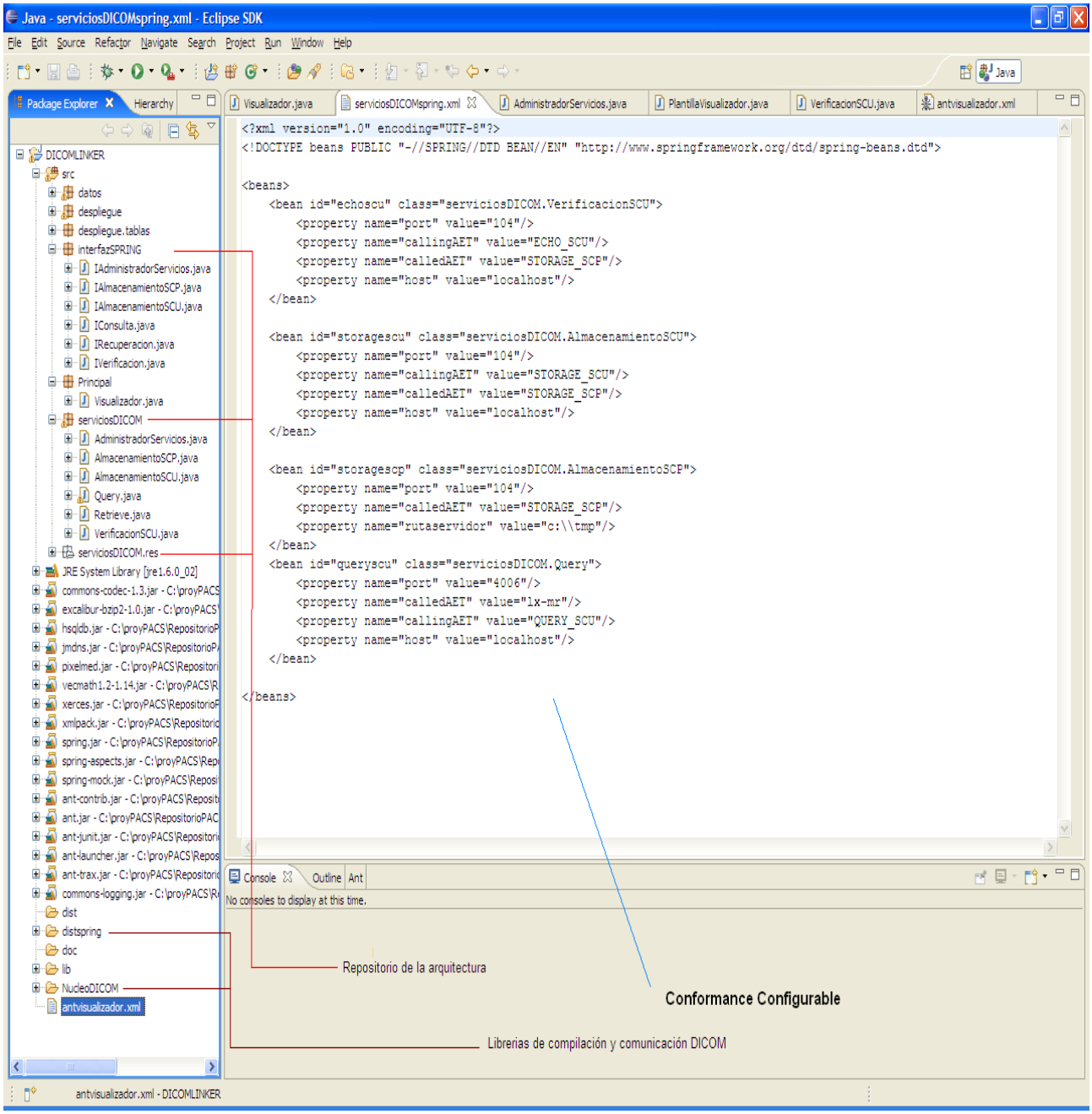

Figura 48: Entorno de desarrollo y configuración para el prototipo.

En la figura 48 también se muestran las partes que conforman el repositorio, las librerías y el *conformance* configurable para este ejercicio.

5.2 Producto final.

A continuación se presentan algunas pantallas del prototipo obtenido para desplegar imágenes DICOM, así como los servicios DICOM configurados en el *conformance* configurable.

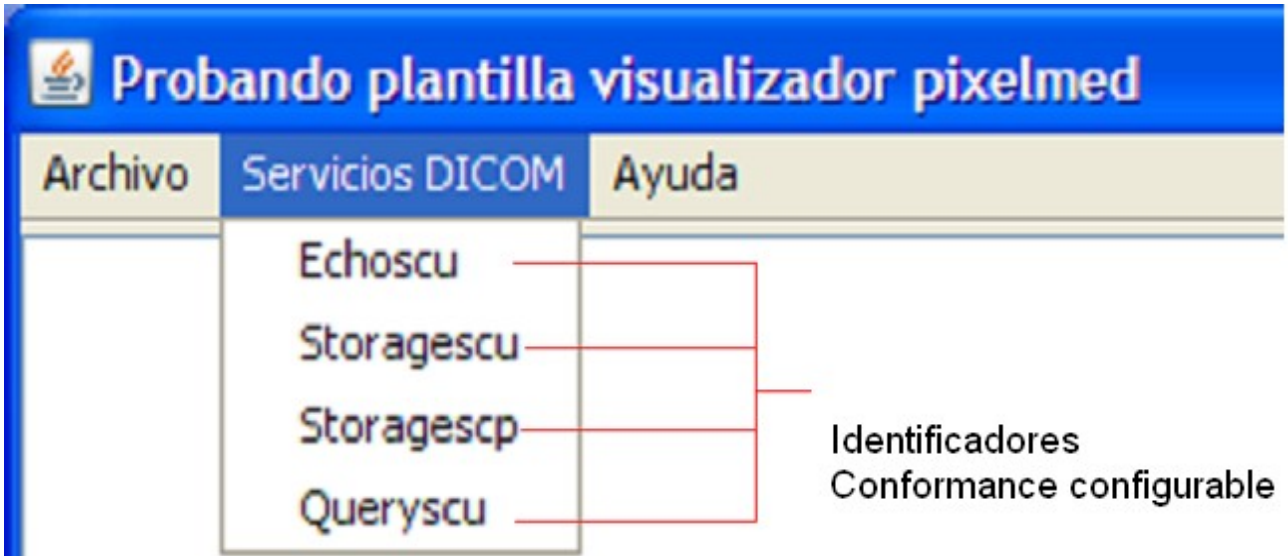

Figura 49: Menú de opciones que la entidad de aplicación registra.

En la figura 49 se observa una ventana que contiene los nombres de los servicios DICOM que se especificaron en el *conformance* configurable: identificadores de las clases de servicio que proporcionan los servicios DICOM para el prototipo de visualización. Para registrar los servicios en cualquier producto solo se requiere el fragmento de código mostrado en la figura 50.

```
/*
ATENCION implementar este método para que el visualizador agregue los servicios
a un menú de opciones
*/
public void AgregaServiciosDICOM(AdministradorServicios serviciosdicom){
 serviciosdicomsoportados=serviciosdicom;
 for(int x=0;x<serviciosdicom.getServiciosRegistrados().length;x++){
   JMenuItem jMenuItem = new
JMenuItem(serviciosdicom.getServiciosRegistrados()[x]); //ELEMENTOS DE
SERVICIO
 jMenuItem.addActionListener(new ejecucionServicioListener());
 jMenu3.add(jMenuItem); //se agregan los menus
 }
}
```

Figura 50: Código para registrar los servicios configurados para el prototipo.

La línea de código en negrita es responsable de solicitar al administrador de *servicios DICOM (AdministradorServicios)* el identificador de cada servicio que se definió en el

conformance configurable.

En la figura 51 se presenta una ventana de diálogo indicando que la aplicación está lista para almacenar imágenes médicas si otra entidad de aplicación desea hacer uso de este servicio por medio del puerto 104.

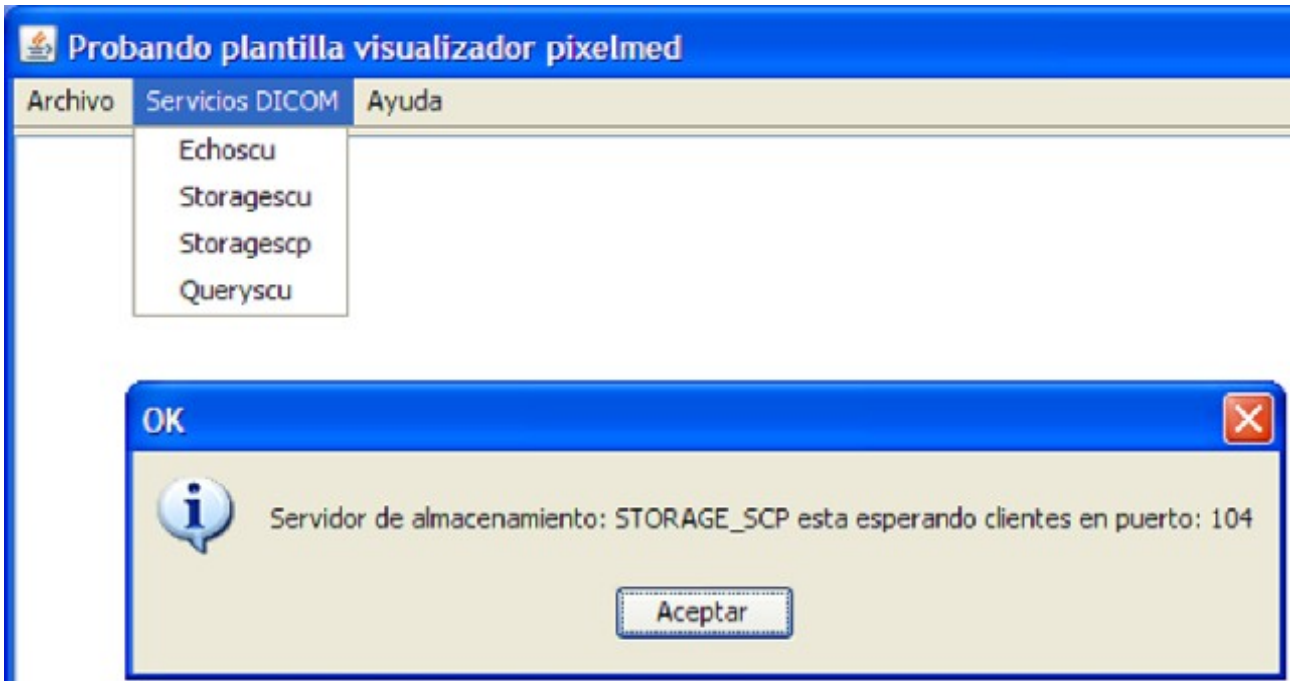

Figura 51: Ejecución de servicio STORAGE_SCP.

Finalmente en la figura 52 se presenta una imagen de tomografía computada con el objetivo de cumplir con lo especificado en la etapa de requerimientos para el producto de visualización.

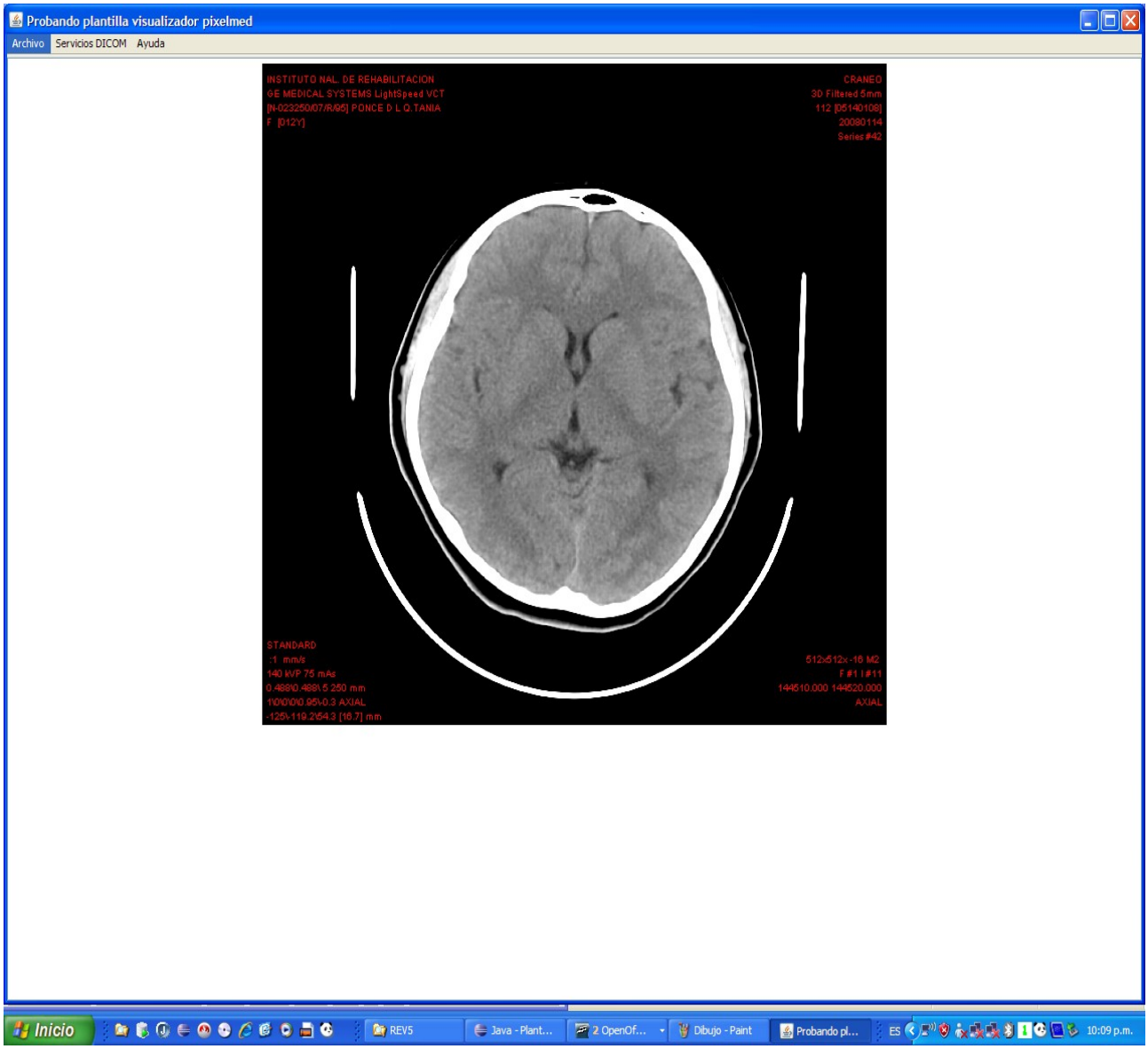

Figura 52: Corrida del prototipo visualizador de imágenes médicas.

El prototipo, por una parte permite visualizar imágenes médicas y por otra proporciona servicios DICOM de verificación en un rol SCU, almacenamiento en ambos roles y consulta en un rol SCU. La deficiencia de este prototipo es que no proporciona un servicio de verificación como proveedor, evitando que otra entidad de aplicación pueda verificar si está activa y tampoco proporciona el servicio de recuperación: puede consultar los estudios de otra entidad de aplicación pero no recuperarlos, a pesar de que implementa el almacenamiento como proveedor. No obstante sería fácil para el desarrollador agregar un par de líneas al *conformance* configurable para que el sistema proporcione estos servicios y hacer más robusto al producto.

5.3 Vista general de la arquitectura del producto visualizador.

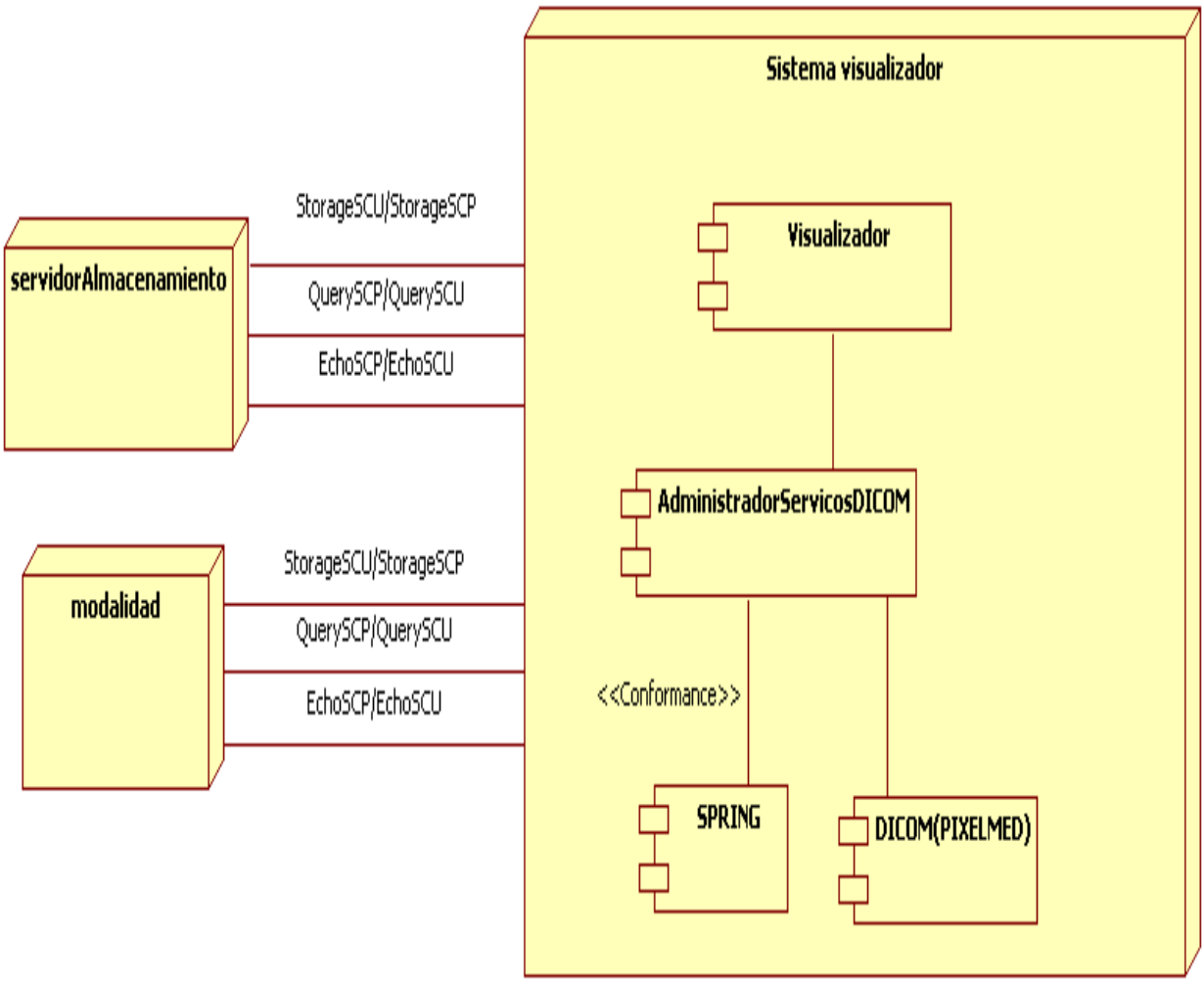

Figura 53: Clasificación de los componentes de la arquitectura del prototipo.

Como se muestra en la figura 53, todo producto generado a partir de la arquitectura de línea de producto se construye con los siguientes componentes:

- Componente del núcleo (*Spring*, DICOM): este componente encapsula las librerías necesarias para generar cualquier producto (core assets) *Spring, Pixelmed*, Interfaces de servicio.
- Componentes particulares al producto (AdministradorServiciosDICOM): este componente siempre implementa las interfaces correspondientes a los servicios DICOM para el producto particular. Lo que es variable para cada producto, son el conjunto de servicios DICOM definidos en el *conformance* configurable.

- Aplicación: es una entidad de aplicación que soporta intercambio de objetos de información DICOM con otra entidad de aplicación, con un comportamiento funcional de acuerdo al objetivo del producto, en este caso visualización de imágenes.

Para agilizar el desarrollo en una línea de producto una de las alternativas es crear un entorno de desarrollo con control de versiones, lo que facilita a cada programador obtener una rama de código en particular. Para este proyecto de tesis sería ideal generar una versión básica para cada producto (Visualización, Almacenamiento, *Workflow*) de PACS. Con lo anterior cada programador cuando requiera implementar un producto particular, deberá hacer una petición al servidor de control de versiones del producto requerido.

5.4 Síntesis del capítulo.

En esta sección se hace una descripción completa de las actividades que se deben realizar para obtener un producto de visualización.

Las dificultades encontradas se deben principalmente a la complejidad de un PACS por lo que fue necesario plantear un proceso de desarrollo que le permita al programador conocer los requerimientos de los diferentes productos. En este sentido, se valida la arquitectura generando un visualizador y resaltando los componentes dedicados a resolver los servicios de comunicación DICOM. No obstante, cada producto puede tener otros componentes: base de datos, procesamiento de imágenes, etc. Sin embargo, debe considerarse que estos componentes se pueden integrar con base en los requerimientos funcionales de la aplicación particular y el mecanismo es semejante al utilizado en el prototipo anterior.

Las grandes ventajas de esta arquitectura se enuncian en los siguientes puntos:

1. La arquitectura toma el control del producto al inyectarle las dependencias de los servicios DICOM, liberando al desarrollador de la complejidad que implica el entender este estándar de comunicación.
2. La arquitectura permite cambiar librerías de forma sencilla para el desarrollador (debido a la implementación de interfaces y el uso de patrones de diseño).
3. El desarrollador solo debe implementar un *conformance* configurable y hacer la

llamada al servicio adecuado, facilitando la modificabilidad del producto.

4. Se acelera la generación de productos requeridos por un PACS particular.

6 Conclusiones y perspectivas.

A continuación se mencionan los aspectos necesarios para concluir con el trabajo de implementar una arquitectura de línea de producto para PACS.

6.1 Discusión crítica.

La implementación de una arquitectura de línea de producto para PACS no es una tarea trivial, ya que se requieren conocimientos de varias disciplinas de las ciencias de la computación y la informática médica. Se debe tener noción sobre: estándares de comunicación, sistemas de almacenamiento de información, sistemas de visualización de imágenes, entre otros; esto con el fin de generar productos que satisfagan las necesidades de una organización. En particular, para este trabajo fue muy importante tomar en consideración los siguientes elementos de discusión, para el desarrollo de la arquitectura para línea de productos PACS.

6.1.1 Desarrollo de la arquitectura para línea de producto PACS.

- Metodologías: el uso de metodologías resulta indispensable para lograr el desarrollo de una arquitectura que soporte todos los requerimientos de un producto para PACS, definidos por los diferentes involucrados de las áreas relacionadas, en una organización dedicada al cuidado de la salud. También, es importante definir una metodología exclusiva para construir productos para PACS, a partir de una arquitectura para línea de producto, que permita al desarrollador centrarse en la identificación del tipo de producto y los componentes de la arquitectura que deberán ser integrados para generarlo y revestirla con la funcionalidad requerida (ver sección 5.1).
- Desarrollo de aplicaciones PACS a partir de la arquitectura de línea de producto: para soportar los atributos de calidad de portabilidad, funcionalidad, Modificabilidad/Extensibilidad en el diseño de la arquitectura para línea de producto, se definieron dos capas que se pueden describir en cuatro niveles de abstracción que facilitan al desarrollador el uso de la arquitectura de línea de producto, para generar nuevas aplicaciones PACS requeridas. Los tres primeros niveles corresponden a los atributos de calidad planteados para la arquitectura para línea de producto y el cuarto nivel soporta la lógica de negocio de la

organización y los requerimientos funcionales. (ver Figura 54).

A continuación se resumen estos niveles:

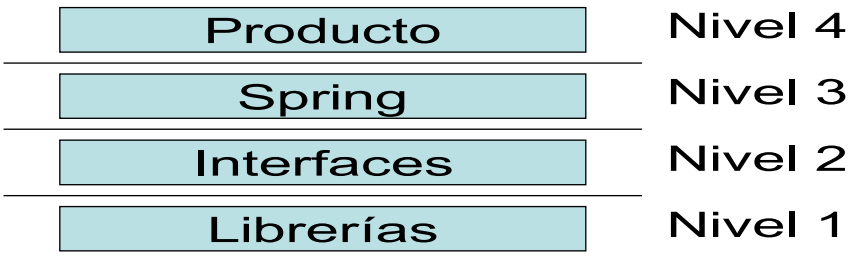

Figura 54: Niveles de abstracción para el núcleo de la arquitectura de línea de producto.

- Primer nivel: representa las librerías DICOM, con todos los servicios estandarizados para implementar cualquier aplicación necesaria para un PACS.
- Segundo nivel: representa un conjunto de interfaces que permiten hacer un uso flexible de las librerías del cuarto nivel, para configurar los servicios ofrecidos por la librería y requeridos por una aplicación particular de un PACS; esto permite configurar en forma flexible los servicios requeridos por la aplicación.
- Tercer nivel: representa el mecanismo necesario para implementar el c*onformance* configurable expresado a través de XML, para una aplicación particular de un PACS. El c*onformance* se utiliza para empaquetar el conjunto de servicios DICOM requeridos a través de un administrador de servicios. De esta forma, el mecanismo ofrece flexibilidad para agregar los componentes necesarios que incluye los servicios DICOM a través de una descripción textual. La descripción es interpretada por S*pring* que ejecuta el mecanismo de inyección de dependencias a través del cual se genera una aplicación PACS definida por el c*onformance*. Esta secuencia de mecanismos es un aporte novedoso para los desarrolladores, ya que permite alejarse de la implementación de los estándares y herramientas empleadas, acelerando el desarrollo de aplicaciones que formarán parte de un PACS.
- Cuarto nivel: los requerimientos funcionales, las restricciones técnicas y las reglas de negocio para el producto, se resuelven en este nivel. En consecuencia, en la propuesta metodológica para la arquitectura de línea de producto se consideró la etapa de captura de requerimientos para que en este

nivel se implemente la funcionalidad que se requiere para una aplicación desarrollada con estándares de calidad que conjugue la integración de NFR,s y FR´s (ver sección 5.2).

- Herramientas y plataformas de desarrollo: es importante considerar las plataformas de desarrollo y herramientas que facilitan la implementación de un sistema. En particular, en este proyecto se decidió utilizar Java como lenguaje de desarrollo para cumplir con el requerimiento no funcional de portabilidad. Dicho en otras palabras, el código es funcional en cualquier sistema operativo que tenga instalada la máquina virtual de Java. Este es un aspecto muy importante en el desarrollo de aplicaciones médicas, ya que se requiere que operen en diferentes plataformas.

  Eclipse, como entorno de desarrollo, facilita la integración natural de otras herramientas importantes como lo son S*pring* y A*nt*. Esta última ofrece mecanismos de compilación separada y organiza e integra los diferentes artefactos involucrados en un proyecto. De esta forma, cualquier modificación es controlada por ant, quien genera una última versión de todos los artefactos involucrados. Por otro lado, si se cambiara de entorno de desarrollo estas herramientas se pueden integrar en forma natural al nuevo entorno.

- Estándares: los estándares son acuerdos o reglas que definen un modelo abstracto, sin embargo, no especifican mecanismos de implementación y son difíciles de entender, además que, como el caso de DICOM, sus especificaciones integran una gran cantidad de información organizada en varios tomos lo cual dificulta su interpretación e implementación. Con el mecanismo empleado en este trabajo se permite la integración de librerías que implementan varias partes del estándar DICOM, como el caso de *Pixelmed*. Esto facilita la integración de los servicios DICOM requeridos por cualquier aplicación PACS, sin tener la necesidad de ser un experto en el estándar DICOM.

- También es necesario considerar el estándar HL7 en el desarrollo de PACS. Este estándar es utilizado en la integración con otros sistemas de información relacionados en el área médica. En la siguiente sección se explican las necesidades de integrar HL7 dentro de la arquitectura de línea de producto para PACS.

6.1.2 Complementos a la arquitectura del núcleo.

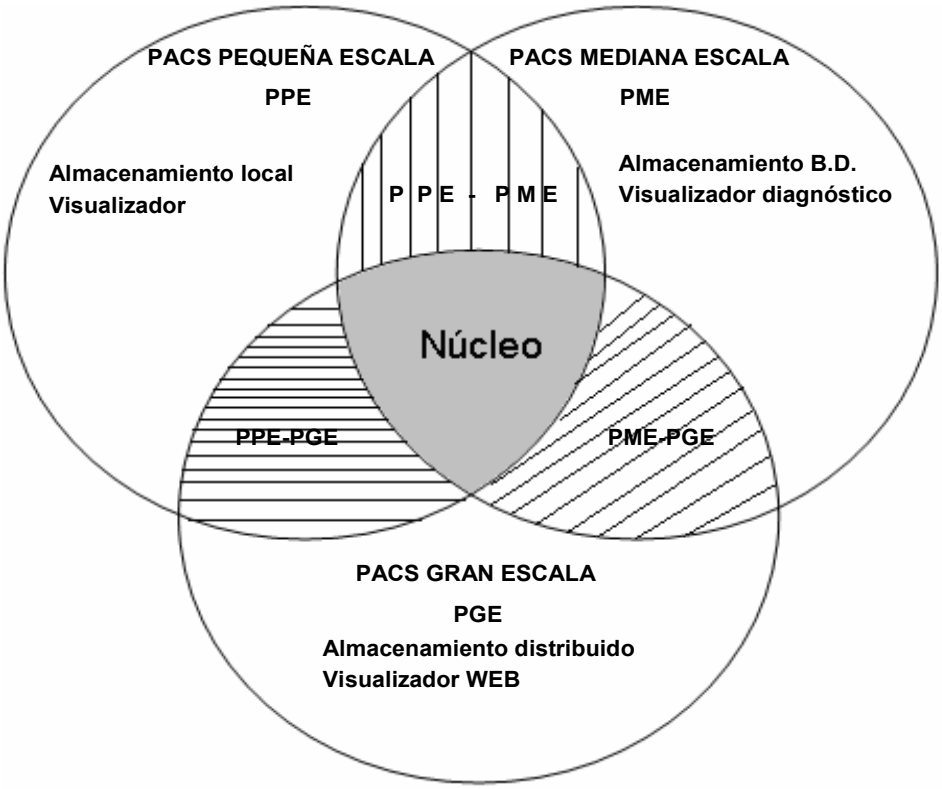

Figura 55: Componentes complementarios para productos de PACS en una línea de producto.

En este trabajo se planteó una solución para el núcleo común de la arquitectura para línea de productos PACS. La figura 55, muestra que un PACS puede contener los mismos productos pero con características diferentes, los productos se pueden construir a partir del núcleo, también se observan intersecciones entre los productos, lo cual implica que deben compartir componentes que faciliten el ensamblado de aplicaciones PACS. Para desarrollar los componentes de las intersecciones, se pueden reutilizar varias de las ideas y mecanismos de forma similar al planteado para el núcleo, sin perder de vista los requerimientos no funcionales para complementar la arquitectura del núcleo.

Después de analizar lo anterior, se plantean un conjunto de componentes candidatos, organizados en diferentes niveles de abstracción, en forma similar a los planteados para el núcleo. Cabe aclarar que para los nuevos componentes podrían requerirse otros servicios DICOM no considerados en el núcleo, así como otras herramientas y metodologías para su implementación.

- Componente de almacenamiento: este componente debe resolver el

almacenamiento en medios físicos o en un sistema de administración de base de datos.

- Componente de acceso remoto: este componente debe proporcionar acceso a la información vía una WAN, interactuando con los servicios DICOM de acceso a la información a través de la WEB (WADO). Al mismo tiempo, se deben ofrecer los servicios de consulta y recuperación de información de la base de datos central del PACS, habilitando un servidor de aplicaciones WEB que ofrezca el servicio a clientes ligeros a través de navegadores WEB.
- Componentes de seguridad: es responsable de asegurar la disponibilidad e integridad de la información, utilizando canales y protocolos de comunicación seguros. También, es indispensable considerar el desarrollo de un mecanismo de validación de usuarios con determinados perfiles de acceso a la información, con el objetivo de que la información particular esté disponible únicamente a usuarios propietarios.
- Componentes para la traducción de mensajes: estos componentes deben convertir peticiones de intercambio de información entre varios sistemas de un hospital: PACS, HIS o RIS. Esto requiere de mecanismos eficientes y eficaces de comunicación entre estos sistemas. Así mismo, se deben considerar los estándares involucrados en la comunicación; dentro de ellos se encuentran: HL7, SQL y DICOM.

El modelo propuesto permite cumplir con los requerimientos no funcionales que complementan la arquitectura del núcleo para la línea de producto. También deben considerarse las restricciones técnicas que se identifiquen cuando se documenten los requerimientos funcionales que son indispensables para un PACS.

A manera de ejemplo: para los productos de un PACS a pequeña escala, quizás funcione sin problemas el mecanismo de persistencia ofrecido por *hibernate* haciendo transparente el diseño de la base de datos a los desarrolladores. Sin embargo, para un sistema a gran escala en donde existen varios niveles de almacenamiento, varios usuarios conectados simultáneamente, la mejor alternativa podrían ser los mecanismos de JDBC de Java para interfazar manejadores como *Oracle* o *Informix*. Esta decisión debe considerarse de acuerdo a las políticas de la organización. No obstante los componentes antes

mencionados deben ser independientes de las restricciones técnicas.

Por otro lado, para los productos de un PACS a mediana o gran escala es necesario considerar también otros estándares de comunicación, como los mencionados anteriormente, lo cual implica hacer un análisis detallado de éstos para proponer el diseño o la inclusión de nuevas librerías.

Es necesario considerar metodologías alternas de desarrollo de sistemas para responder eficientemente a las necesidades de implementar componentes con varias tecnologías, como lo son bases de datos, aplicaciones web y estándares como HL7.

6.1.3 Elementos formativos para el desarrollo profesional.

Dentro de la experiencia personal para la práctica profesional, se puede destacar lo siguiente:

- Al médico radiólogo no le interesa una aplicación diseñada con metodologías y estructuras arquitectónicas, solo piensa en una aplicación funcional, de ser posible de un fabricante reconocido. De esta forma, se requiere generar aplicaciones de forma rápida que puedan competir con las diseñadas comercialmente. En este sentido, una arquitectura de línea de producto para PACS es ideal ya que se genera a partir de un núcleo e intersecciones y cada producto puede agregar funcionalidad complementaria para hacer crecer el sistema eficientemente de acuerdo a la funcionalidad requerida. Así los productos generados pueden competir con los productos comerciales.
- Aprender a identificar fronteras para el desarrollo de sistemas utilizando metodologías que permiten implementar productos con calidad.
- Aprender a desarrollar sistemas muy complejos como los PACS, desde otra perspectiva a partir de un modelo genérico representando una línea de producto.
- La posibilidad de proponer un diseño que permite generar productos PACS sin conocer a fondo varios detalles, como las especificaciones del estándar DICOM.

6.2 Conclusiones.

Aplicando un proceso de desarrollo de software y representando los modelos en UML ha sido posible obtener diferentes modelos que representan el núcleo para la arquitectura de

la línea de producto a través de un conjunto de vistas estáticas y dinámicas.

La metodología propuesta en este proyecto fue adecuada para identificar los elementos importantes del núcleo y los mecanismos de ensamblado para los productos de un PACS, sin embargo se requiere realizar otro análisis para determinar cuáles son las herramientas y metodologías para implementar los componentes de intersección que se plantearon a través del modelo respectivo.

Debido a la complejidad de los PACS, fue necesario crear un proceso de desarrollo que facilite a los desarrolladores la implementación de los productos que se deben generar a partir de la arquitectura. La metodología propuesta, nos ayuda a identificar las restricciones técnicas que impone la organización, ya que se contempla una etapa de captura de requerimientos debido a que los PACS no son sistemas genéricos, es decir, deben apegarse a las necesidades de la organización y en este sentido nuestra propuesta tiene una gran ventaja sobre los productos comerciales, ya que no existen dependencias sobre un sistema o una versión especifica de alguna aplicación para el funcionamiento de los productos.

Otro punto importante a destacar es que la arquitectura esta pensada para desarrolladores que no tengan un conocimiento detallado del estándar DICOM. Esto es un logro muy especial ya que con conocimientos de patrones de diseño y compilación condicional, se diseña una aplicación que cumpla con las especificaciones de un PACS, sin tener que conocer la funcionalidad interna de la implementación del estándar DICOM. Con solo analizar la parte 2 (DICOM *conformance*) del estándar el desarrollador tiene una mecanismo para realizar la abstracción general del producto deseado y la arquitectura se encarga de ensamblarlo. El proponer un mecanismo con diferentes niveles de abstracción generó un conjunto de componentes con alta cohesión independientes que facilitan la integración de librerías o nuevos componentes.

6.3 Perspectivas.

Es necesario desarrollar todos los componentes de intersección mencionados para automatizar el flujo de trabajo en las áreas de radiología de un hospital, así como los servicios complementarios y necesarios que dan seguimiento al diagnóstico final del paciente (interfaces hacia HIS-RIS-PACS). Así mismo, los servicios de consulta remota

para ofrecer a un hospital de 3er nivel la implementación de servicios de “teleradiología” que cumplan otras necesidades. De esta forma se obtendrá una línea de producto exitosa para generar productos para PACS que compitan con cualquier producto comercial.

Para dar continuidad a este proyecto es necesario generar un caso de prueba real, es decir, se recomienda generar un vínculo con un hospital que requiera de un PACS y de esta forma obtener los requerimientos por medio del proceso propuesto en el capítulo 3, posteriormente configurar su propio *conformance* con todos los productos del PACS. Este proyecto relacionado con el sector salud permite ofrecer alternativas de solución viables, acorde a las necesidades particulares de cada institución.

Apéndice A. Siglas y abreviaturas

ACR	American College of Radiology
API	Abstract Programming Interface
C-ECHO	Composite-ECHO
C-FIND	Composite-FIND
C-GET	Composite-GET
C-MOVE	Composite-MOVE
C-STORE	Composite-STORE
CR	Computed Radiography
CT	Computed Tomography
DICOM	Digital Image Communication in Medicine
AE	Aplication Entity
HIS	Hospital Information Systems
HL7	Health Level 7
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers
IOD	Information Object Definition
ISO	International Organization for Standarization
MR	Magnetic Resonance
NFR	Non Functional Requirements
NEMA	National Electrical Manufacturers Association
NM	Nuclear Medecine
PACS	Picture Archiving and Communications Systems
RIS	Radiology Information Systems
RUP	Rational Unified Process
SEI	Software Engineering Institute
SC	Secondary Capture
SCM	Software Configuration Magement
SCP	Service Class Provider
SCU	Service Class User
SOP	Service Object Pair
SPL	Software Product Line
SQL	Structured Query Language
UML	Unified Modeling Language
US	UltraSound
WADO	Web Access to DICOM Persistent Objects
XML	Extensible Markup Language

Referencias.

[ANT2007] Página web de Stephane Bailliez, Nicola Ken Barozzi, “et al.”, “Manual de Apache Ant”, [en línea] <http://ant.apache.org/manual/index.html> [consulta:Septiembre 2007]

[Bass2003] Bass, L., Clements, P. y Kazman, R., “Software Architecture in Practice”, 2d edition. Addison Wesley, 2003.

[Booch1998] Grady Booch, “Leaving Kansas”, IEEE Software 15 (1), Jan-Feb 1998, pp 32-35.

[Booch1999] Ivar Jacobson, Grady Booch, James Rumbaungh, “The Unified Software Development Process” Addison-Wesley 1999.

[Buschmann1996] Buschmann F., Meunier R., Robnert H., Sommerland P., Stal M., “Patterns-Oriented Software Architecture, A System of Patterns” Volume 1, John Wiley & Sons, 1996.

[DICOM2008] Página web de “Digital Imaging and Comunications in Medicine (DICOM)", [en línea] <ftp://medical.nema.org/medical/dicom/2008> [consulta: Enero 2008]

[ECLIPSE2007] Página web de “Eclipse Modeling Framework”, [en línea] <http://www.eclipse.org/modeling/emf/> [consulta: Septiembre 2007]

[Forouzan2004] Behrouz A. Foruzan. “Data Communications and NetWorking”, 3d edition. Mc. Graw Hill, 2004.

[Gamma1995] Erich Gamma, Helm R.,Johnson, R.; & Vlissides, J. “Design Patterns: Elements of Reusable Object-Oriented Software”, ADDISON-WESLEY, 1995.

[GammaEO1995] Erich Gamma and Richard Helm. “Designing Objects for Extensions”, In Dr. Dobbs Sourcebook, #236 pages 56-59, May/June 95.

[HKHuang2004] H. K. Huang, D.Sc., FRCR (Hon.), “PACS And Imaging Informatics Basic Principles And Applications.”, Wiley & Sons, Inc. 2004.

[IanGorton2006] Ian Gorton, “Essential Software Architectura” Springer-Verlang, 2006.

[IBM2007] Página web de Jim Heumann, “Introduction to business modeling using the Unified Modeling Language (UML)”, [en línea] <http://www-128.ibm.com/developerworks/rational/library/360.html> [consulta: Agosto 2007]

[IEEE10612007] Página web de Mario R. Barbacci, “Software Quality Attributes: Modifiability and Usability”, Software Engineering Institute, Carnegie Mellon University, [en línea] <http://www.ieee.org.ar/downloads/Barbacci-05-notas1.pdf> [consulta: Agosto 2007]

[IHE322006] “International Hospital Equipment & Solutions”, February/March 2006, Volume 32, No. 1. page 24.

[IPI2003] HK Huang,“Image Processing And Informatics Laboratory”, Deparment of Radiology Childrens Hospital Los Angeles, 2003 Annual Report, University of Southern California.

[ISO91262007] Página web de Geoffrey S. Hubona. “Georgia State University”, [en línea] <http://www.cis.gsu.edu/~ghubona/cis8300/ISO9126.pdf> [consulta: Agosto 2007]

[Keith2006] Keith J. Dreyer, David S. Hirschor, ET. AL. “PACS a Guide To Digital Revolution”, 2d edition, Springer-Verlang Berlin Heidelberg. 2006

[Kruchten1999] Philippe B. Kruchten, “The 4+1 View Model of Architecture”, IEEE SOFTWARE NOVEMBER 1995, pp. 42-50.

[Kruchten2004] Philippe Kruchten, “The Rational Unified Process and Introduction”, Third Edition Addison-Wesley, 2004.

[Leotta1993] D. F. Leotta, Y. Kim. “Requirements for Picture Archiving and Communications.” IEEE ENGINEERING IN MEDICINE AND BIOLOGY March 1993 pp. 62-69.

[Oleg2008] Oleg S. Pianykh. “Digital Imaging and Communications in Medicine (DICOM) A Practical Introduction and Survival Guide”, Springer-Verlang Berlin Heidelberg. 2008

[Osterweil1987] L. Osterweil, “Software processes are software too”, Proceedings of the 9th international conference on software Engineering (ICSE). 1987.

[PIXELMED2007] Página web de David Clunie´s, librerías de comunicación DICOM ”PIXELMED”, [en línea] <http://www.dclunie.com/pixelmed/software/> [consulta: Octubre 2007]

[RUPDataSheet2007] Página web de “IBM”, [en línea]

<ftp://ftp.software.ibm.com/software/rational/web/datasheets/RUP_DS.pdf> [consulta: Octubre 2007]

[Sajeesh2008] Sajeesh Kumar, “Teleradiology”, Springer-Verlang Berlin Heidelberg. 2008

[SEISPLFW2008] Página web de sei.cmu.edu, “SEI”, [en línea] <http://www.sei.cmu.edu/productlines/frame_report/PL.essential.act.htm> [consulta Enero 2008]

[SEIDescriptions2008] Página web de sei.cmu.edu, “SEI”, [en línea] <http://www.sei.cmu.edu/architecture/start/reasoning.cfm> [consulta Enero 2008]

[Shortliffe2001] Shortliffe E., “et al.”, “Medical Informatics. Computer Applications in Health Care and Biomedicine”, 2d edition, Springer. 2001.

[Siegel2002] Siegel E., Huang H., “PACS and integrated medical information system: design and evaluation”, Progress in biomedical optics and imaging, vol 3, No. 23, ISSN 1605-7422.

[SPRING2008] Página web de springsource.org. “Java Application Framework”, [en línea] <http://www.springframework.org/> [consulta: Enero 2008]

[SWEBOK2004] Página web de SWEBOOK IEEE Computer Society. “Guide to the Software Engineering Body of Knowledge – 2004”, [en línea]. <http://www.computer.org/portal/web/swebok/html/ch1> [consulta:
Abril 2010]

[Wiegers2003] Karl E. Wiegers, “Software Requirements”, 2d edition. Microsoft Press, 2003.

Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA- IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

DISEÑO DE UN NÚCLEO DE ARQUITECTURA COMÚN PARA CONSTRUIR PRODUCTOS DE PACS.

Tesis que presenta:
Marco Antonio Núñez Gaona.

Para obtener el grado de:
Maestro en Ciencias y Tecnologías de la Información.

Asesores:

Dr. Humberto Cervantes Maceda.

M. en C. Alfonso Martínez Martínez.

Jurado Calificador:

Presidente: Dr. Juan Ramón Jiménez Alanís.

Secretario: Dra. Perla Inés Velasco Elizondo.

Vocal: Dr. Humberto Cervantes Maceda.

México, D. F. Julio 2010.