Protocolo de Servicio de Intercambio de Mensajes DICOM

Juan Ramón Jiménez Alaniz

Maestría en Ciencias
(Ingeniería Biomédica)

Casa abierta al tiempo

UNIVERSIDAD AUTONOMA METROPOLITANA

Iztapalapa

C. B. I.

Asesora: Verónica Medina Bañuelos
Co-Asesor: Alfonso Martínez Martínez

México, D. F. a 21 de junio del 2000

PROTOCOLO DE SERVICIO DE INTERCAMBIO DE MENSAJES DICOM

PROTOCOLO DE SERVICIO DE INTERCAMBIO DE MENSAJES DICOM

1. Introducción

1.1. Problema del manejo de imágenes en hospitales

El uso de la tecnología en imágenes para diagnóstico tiene gran importancia en los sistemas de salud modernos. El volumen de las imágenes médicas que se generan en el ambiente clínico ha crecido rápidamente debido tanto al aumento del número de exámenes que se realizan como al rango de modalidades disponibles. Aunque la imagen de rayos X es la más común, en los exámenes clínicos ahora se incluye la tomografía computarizada (CT), la resonancia magnética (MR), el ultrasonido y la medicina nuclear (NM), entre otras. Mientras que un examen de rayos X comprende de una a cuatro vistas, un examen típico de CT o MRI incluye de 20 a 100 imágenes representando secciones transversales de alguna parte del cuerpo del paciente. Para un hospital de 600 camas, anualmente se tienen aproximadamente un millón de imágenes, lo cual requiere cerca de 2 TB ($2X10^{12}$ bytes) de espacio de almacenamiento. Esta cantidad crecerá a medida que se vuelvan más comunes los estudios de CT y MRI, que los equipos de imágenes se usen con más frecuencia en los hospitales y que se acepten nuevas modalidades de imágenes.

Este volumen de imágenes claramente representa un reto para la adquisición de datos, administración, distribución y almacenamiento. Una alternativa es el manejo de imágenes a través de dispositivos conectados en red, que en conjunto ofrezcan una serie de servicios que den soporte operacional a un área radiológica, ofreciendo además, facilidad, rapidez y seguridad en el acceso de las imágenes, y calidad en su representación.

1.2. Sistemas PACS

El desarrollo de nuevas técnicas en imágenes y el incremento de modalidades de imágenes que las generan en formato digital lleva de forma natural al desarrollo de sistemas de administración de imágenes digitales. Tales sistemas referidos como Sistemas de Comunicación y Archivo de Imágenes (PACS por sus siglas en inglés "Picture Archiving and Communication System") están surgiendo en ambientes clínicos y radiológicos. Los primeros diseños de PACS fueron sistemas centralizados donde las imágenes y los datos se almacenaban en un archivo central para ser accesados por peticiones de estaciones de trabajo periféricas. Con el desarrollo de las redes de computadoras ha sido posible diseñar archivos distribuidos, donde las imágenes y datos se almacenan en lugares distintos de la red, siendo accesibles en cualquier parte de la red.

La evolución en el diseño de PACS avanza hacia una arquitectura abierta, con equipo de distintos proveedores basado en estándares de la industria para interconexión de dispositivos de imágenes y otros componentes PACS. Esta tendencia es posible por una evolución similar entre los fabricantes de equipo médico, cuya tendencia es soportar los estándares establecidos y ser menos protectores en cuanto a la integración de sus equipos con otros, permitiendo que los datos de imágenes se transfieran a otros dispositivos o computadoras. Esto permite ser más realista en un ambiente clínico, donde los diferentes equipos para adquisición de imágenes raramente se le compran a un solo vendedor, y en muchos casos, los departamentos de radiología están equipados con equipo heterogéneo de diferentes marcas.

Uno de los primeros objetivos de un PACS es coleccionar imágenes radiológicas en formato digital para archivo y comunicación. Aunque hay un gran número de imágenes radiológicas que se adquieren directamente en formato digital (CT, MRI, medicina nuclear, etc.), muchos fabricantes no dan un fácil acceso a las imágenes en su formato digital original. A finales de los ochentas han surgido algunos estándares de la industria que permiten que las imágenes se transfieran desde los dispositivos de adquisición a los ambientes PACS. Muchos proyectos de PACS se han enfrentado con la dificultad de extraer las imágenes de los dispositivos de adquisición, para lo que se han desarrollado interfases a la medida, que son muy costosas porque las soluciones individuales son siempre más caras que las genéricas que se pueden reusar en diferentes configuraciones. Por la misma razón es muy difícil integrar equipo de imágenes de diferentes fabricantes y algunos proyectos de PACS se han limitado a equipo de un sólo fabricante. Tal solución es inaceptable en hospitales grandes donde se requiere de equipo de imágenes actualizado que sólo se puede obtener seleccionando el mejor equipo de fabricantes diferentes.

1.3. Estándares: ACR-NEMA, DICOM

Al inicio de los PACS, muchos grupos cuyo trabajo estaba relacionado con los PACS encontraron el problema de adquirir imágenes y datos relacionados de equipos de diferentes fabricantes. Este problema se discutió entre la comunidad radiológica a través del Colegio Americano de Radiología (ACR), y los fabricantes también empezaron a darse cuenta que ganarían poco si mantenían un formato propietario. En 1983 el ACR y la Asociación de Fabricantes de Eléctricos Nacional (NEMA, "National Electrical Manufacturers Association") formaron un comité conjunto en un esfuerzo por desarrollar un estándar por medio del cual los usuarios de equipo de imágenes médicas digitales pudieran interconectar equipos diversos. La misión de este grupo, llamado Comité de Estándares en Comunicaciones e Imagen Digital ACR-NEMA fue desarrollar un estándar para:

- Promover la comunicación de información de imágenes digitales, sin importar el fabricante del dispositivo

- Facilitar el desarrollo y expansión de PACS que puedan también interconectarse con otros sistemas de información del hospital

- Permitir la creación de bases de datos de información de diagnóstico que se pueda consultar por una amplia variedad de dispositivos distribuidos geográficamente.

Después de 2 años de trabajo, la primera versión del estándar, ACR-NEMA 300-1985 (también llamada ACR-NEMA Versión 1.0) fue distribuida en la reunión anual de la RSNA (Radiological Society of North America) y publicado por NEMA. En esta primera versión se encontraron errores y se hicieron sugerencias para mejorarlo. Los cambios fueron encargados a un grupo de trabajo y en 1988 se publicó el ACR-NEMA 300-1988 (o ACR-NEMA Versión 2.0). El problema fue que en 1988 muchos usuarios querían interconectar dispositivos de imagen y una red; aunque esto se podía hacer con la versión 2.0, le faltaban partes que fortalecieran la comunicación en red. Ya que la versión 2.0 no había sido diseñada para conectar equipo directamente a la red, resolver este problema significó cambios mayores al estándar. Se decidió que el desarrollo de una interfase para soporte de red requería más que parchar la versión 2.0, por lo que el proceso de diseño total tenía que usar re-ingeniería, y el método adoptado fue el diseño orientado a objetos [Booch94].

Examinando los tipos de servicios necesarios para comunicarse sobre diferentes redes, se definió un servicio básico que permitiera a la capa superior de los procesos de comunicación "hablar" con diferentes protocolos de red. Estos protocolos se modelan como series de capas, con frecuencia referidos como "pilas". La pila de la versión 2.0 se definió como una conexión punto a punto, se eligieron otras dos basándose en la popularidad y la futura expansión: TCP/IP (Transmission Control Protocol/Internet Protocol) e ISO-OSI (International Standards Organization-Open Systems Interconnection). La filosofía de diseño básica fue que, dada una aplicación de imágenes médicas, ésta pudiera comunicarse sobre cualquiera de las pilas con otro dispositivo que usara la misma pila. Con apego al estándar, sería posible cambiar las pilas de comunicación sin tener que reescribir los programas de la aplicación. Después de tres años de trabajo, con muchas sugerencias valiosas de la industria y la academia se completó el ACR-NEMA DICOM (Digital Imaging and Communications in Medicine), también conocido como DICOM \ Versión 3.0.

Dicho de una manera muy simple DICOM es un estándar para la comunicación de imágenes médicas e información relacionada, y que contiene las siguientes mejoras en comparación con las versiones previas del estándar:

- Es aplicable a un ambiente de red. Las versiones previas fueron aplicables a un ambiente punto a punto solamente. DICOM soporta operación en red usando protocolos estándar como OSI y TCP/IP.

- Especifica la manera en que los dispositivos demandan (reclaman) conformidad para que el estándar reaccione al intercambio de comandos y datos.

- Especifica niveles de conformidad, explícitamente describe cómo un implementador debe estructurar una Declaración de Conformidad para seleccionar opciones específicas.

- Está descrito en un documento estructurado en múltiples partes, esto facilita la actualización del estándar en un ambiente de evolución rápida, simplificando la adición de nuevas características.

- Introduce Objetos de Información explícitos no sólo para imágenes y gráficos, sino también para estudios, reportes, etc.

- Especifica una técnica establecida para identificar únicamente cualquier Objeto de Información.

1.4. Problema de diseño de aplicaciones

DICOM es un ambiente o marco de trabajo orientado a objetos en el cual la información (datos) y funciones o rutinas que operan sobre la información (métodos) se agrupan juntos en paquetes fáciles de administrar llamados objetos. La importancia del diseño orientado a objetos es que produce software que es menos complejo y fácil de cambiar que el software producido por la aproximación de procedimientos o funcional.

Revisando algunos desarrollos de software que se han hecho sobre el estándar DICOM, se puede observar que existe un alto grado de dificultad para poder entender la forma o el proceso con el cual han sido desarrollados, y aun se torna más difícil, si no es que imposible, el realizar alguna modificación a una parte de ese software. Como ya se mencionó, DICOM es un estándar que contempla el modelo orientado a objetos, buscando con ello que los desarrollos en software del mismo, sean más rápidos y que se elimine la necesidad de reinventar cada vez que se pretende hacer una implementación de DICOM, dado que depende de modelos explícitos y detallados de cómo están descritos y relacionados los objetos (pacientes, imágenes, reportes, etc.) involucrados en operaciones de radiología. Estos modelos llamados modelos entidad-relación, son una manera de asegurar que fabricantes y usuarios entiendan las bases para desarrollar estructuras de datos usadas en DICOM. Esta aproximación para desarrollar estructuras de datos basadas en modelos y análisis de entidades reales abstraídas, usadas en el

modelo, es el diseño orientado a objetos. El diseño orientado a objetos también proporciona una manera de describir no sólo la información sino que hacer con la información, o como los programas pudieran accesar la información relacionado con una colección de objetos. Los métodos están asociados con los objetos definidos, DICOM hace uso de este concepto definiendo servicios como "almacenar una imagen" u "obtener información de paciente". Estos servicios se implementan en DICOM usando construcciones conocidas como un conjunto genérico de operaciones y notificaciones llamadas Elementos de Servicio de mensaje DICOM (DIMSE).

Dado que DICOM usa el modelo de objetos y para evitar el tener que reinventar es necesario aplicar un proceso de desarrollo de software, de tal manera que aumente las posibilidades de obtener un producto de software de alta calidad, robusto y de fácil mantenimiento.

Un proceso define quién esta haciendo qué, cuándo y cómo alcanzar cierta meta. En la ingeniería de software la meta es construir un producto de software o incrementar uno existente. Un proceso efectivo proporciona directivas para el desarrollo eficiente de software de calidad. Captura y presenta las mejores prácticas que permite el estado del arte actual. En consecuencia, reduce el riesgo e incrementa la predecibilidad.

Un proceso de desarrollo de software es un método para organizar las actividades relacionadas a la creación, entrega, y mantenimiento de los sistemas de software. Es el conjunto de actividades necesarias para transformar los requerimientos del usuario en un sistema de software.

Hoy la tendencia del software es hacia sistemas muy grandes y más complejos. Esto se debe en parte al hecho de que las computadoras son cada vez más poderosas llevando a los usuarios a esperar más de ellas. Nuestras demandas de software potente y complejo no se igualan con la forma en que se desarrolla el mismo. Mucha gente desarrolla software usando los mismos métodos de hace 25 años. Esto es un problema, a menos que actualicemos nuestros métodos, no seremos capaces de lograr nuestra meta de desarrollar el software complejo que se necesita hoy.

Esta actualización debe abarcar tanto la aplicación de un proceso de desarrollo, así como las herramientas que permitan modelar la solución del dominio del problema. Esto último se puede lograr utilizando el Lenguaje de Modelado Unificado, UML por sus siglas en inglés, dado que es un lenguaje para especificar, visualizar y construir los artefactos de sistemas de software. Es un sistema notacional (incluyendo semántica) que ayuda en el modelado de sistemas usando conceptos orientados a objetos.

2. Objetivo

- Realizar el análisis, diseño y construcción del protocolo para el Servicio de Intercambio de Mensajes DICOM, conocido como DIMSE.

- Utilizar el modelo de objetos para la elaboración del análisis, diseño y construcción de este proyecto.

- Usar el Lenguaje de Modelado Unificado (UML) para reportar los productos obtenidos en las actividades realizadas en las diferentes fases de la elaboración del proyecto.

- Aplicar la metodología del proceso de desarrollo para proyectos de software propuesta por Craig Larman [Larman98] en la propuesta de diseño del proyecto.

3. Metodología

3.1. Descripción de DICOM

DICOM es un estándar que cubre parcialmente el modelo de procesos distribuidos, el cual tiene al menos dos procesos (socios) compartiendo información, cada uno haciendo su propio proceso pero confiando en la funcionalidad del otro. DICOM usa su propia terminología para describir el contexto, relación, etc. como puede apreciase en la figura 3.1. La relación entre ambos roles (es decir, los procesos) esta definido por la descripción de la Clase Servicio, esta relación define cuál proceso y en qué condición toma la iniciativa. La Clase Servicio describe los papeles de ambos procesos, que en DICOM se llaman: Usuario de la Clase Servicio o SCU (Service Class User), este lado usa la funcionalidad del otro lado y tiene el rol de cliente; y Proveedor de la Clase Servicio o SCP (Service Class Provider), actuando como servidor. Parte de la Clase Servicio es la descripción de la información y operaciones que ambos procesos están de acuerdo en intercambiar. En la información se considera la semántica, no la manera en que ella se representa (sintaxis) y esta definida por el contexto del servicio del proceso distribuido; las operaciones definen cómo debe ser procesada la información intercambiada. En DICOM, la información y operaciones, se combinan en la definición de la clase, y se llama Clase Par Objeto Servicio o SOP (Service Object Pair Class). En cada definición de Clase SOP una sola Definición de Objeto de Información o IOD (Information Object Definition) se combina con uno o más servicios [Revet97].

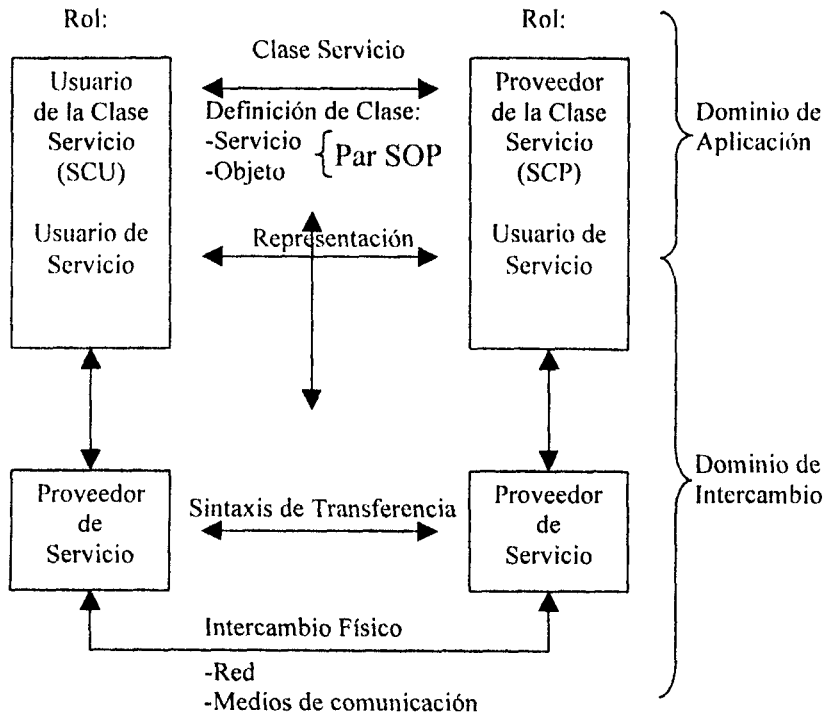

Figura 3.1 Modelo de Proceso Distribuido transformado a DICOM

La combinación de contexto, relación, información y operaciones son parte del dominio de la aplicación del proceso distribuido y se deben determinar antes de realizar una implementación exitosa. Los conceptos anteriores no tienen que ver con la manera en que realmente se intercambia la información, sino que se confía en los servicios de nivel inferior proporcionados por el dominio de intercambio para realizar esta transferencia.

Antes de intercambiar información la identificación de la Clase SOP es un problema importante que tiene que resolverse primero. El mecanismo usado depende del tipo de intercambio: red o medios. Usando la Clase Servicio y otras funciones derivadas, los procesos, en un ambiente de procesamiento distribuido, funcionan juntos por medio de los servicios proporcionados por el dominio de intercambio.

La parte de la información de una clase SOP esta definida en los IODs. Un IOD es una colección de piezas de información relacionadas, agrupadas en Entidades de Información. Cada entidad contiene información acerca de un solo objeto (del mundo real) tal como un paciente, una imagen, etc. Dependiendo del contexto definido por la Clase de Servicio un IOD consiste de una sola entidad de información, llamada IOD normalizado, o una combinación de entidades de información, llamadas IOD compuesto.

Los elementos de servicio son las operaciones permitidas sobre los Objetos de Información para una cierta Clase SOP. El grupo de elementos de servicio perteneciente a la Clase SOP se llama Grupo de Servicio.

El Grupo de Servicio de una clase SOP se selecciona de una lista de Elementos de Servicio DICOM. Algunos Elementos de Servicio tienen el propósito de usarse con un IOD compuesto, otros para usarse con IODs normalizados. Una tercera categoría, Elementos de Servicio relacionados al medio de almacenamiento, manejan instancias de Clases SOP compuestas y normalizadas como archivos.

El contexto descrito por la Clase de Servicio esta limitado cuando se usan IODs compuestos (por ejemplo, transferir una imagen). Tales Elementos de servicio tienen un significado complejo, por ejemplo, Almacenar, Encontrar, Mover. No hay relación supuesta entre Elementos de Servicio individuales en una secuencia cuando se usan Clases de Servicios compuestas. Si existe una relación, esta fuera del alcance de las Clases de Servicio y deberá definirse en el proceso de uso de las Clases de Servicio.

En contraste, las Clases de Servicio usando IODs normalizados tienen un contexto más amplio, tal como administrar funciones. Usan primitivas de Elementos de Servicio para operaciones con piezas simples de información: Obtener, Establecer, Acción, etc. La Clase de Servicio define la relación de una secuencia de peticiones de primitiva. Con las Clases de Servicio normalizadas ambos

procesos, mantienen comunicación con el procesamiento de ambos lados, usando los Elementos de Servicio para controlarlos.

Cada clase SOP usa uno o más Elementos de Servicio ya sea del grupo compuesto (C-XXXX) o del grupo normalizado (N-XXXX). Los siguientes Elementos de Servicio están disponibles: C-STORE, C-FIND, C-MOVE, C-GET, C-CANCEL, C-ECHO, N-GET, N-SET, N-ACTION, N-CREATE, N-DELETE, y N-EVENT-REPORT. La semántica de los Elementos de Servicio depende de la Clase Servicio y Clase SOP en las cuales se usen.

Los Elementos de Servicio relacionados a los medios M-WRITE, M-READ, M-DELETE, M-INQUIRE-FILE-SET y M-INQUIRE-FILE definen funciones primitivas para manipulación con conjuntos de archivos.

El marco de trabajo de las definiciones de arriba toma forma cuando se usan en un proceso distribuido. Después de acordar cuales Clases SOP se soportan (e implícitamente la Clase Servicio), y como están divididos los papeles de SCU y SCP, las instancias de Clase SOP se pueden intercambiar entre las dos contra partes.

Después de colectar la información, será codificada para los formatos DICOM, usando la Etiqueta y Representación de Valor para crear un conjunto de Datos DICOM, en el cual cada atributo se codifica en un elemento de datos. Este conjunto de datos lo maneja el proveedor de servicio de intercambio, el cual asegura que la contra parte recibe un conjunto de datos igual. Las diferencias en representación específica de sistema se toman en cuenta durante el intercambio, asegurando que los valores semánticos permanecen intactos.

El receptor del conjunto de datos extrae la información que necesita y actúa de acuerdo a la semántica de la Clase SOP.

Antes de que el conjunto de datos de una instancia SOP se pueda intercambiar, la manera en que el conjunto de datos esta codificado para un flujo de bytes tiene que quedar establecida por acuerdo ya sea cuando se usa un intercambio de red, o un almacenamiento junto con los datos en un medio. La sintaxis de transferencia define un conjunto de reglas de codificación usadas para representar sin ambigüedad este conjunto de datos.

Tres aspectos tienen que estar definidos por la sintaxis de transferencia:

1. De que manera esta especificada una Representación de Valor, VR. Una VR describe cómo se codifica un atributo en un elemento de datos.
2. El ordenamiento de bytes de números de bytes múltiples (palabras, palabras largas): little endian o big endian.
3. En caso de encapsulamiento (compresión): el formato de compresión.

El manejo de la sintaxis de transferencia es parte del proveedor de servicio. Sin embargo, ambos procesos tienen que iniciar el establecimiento de una sintaxis de transferencia correcta, aceptable para ambos procesos.

3.1.1. Conceptos de red DICOM

Hasta este punto solo se han discutido los conceptos de DICOM del dominio de la aplicación. Cuando se usa una red para intercambiar información, el dominio de intercambio contendrá funciones requeridas para la comunicación: el dominio de comunicación. Este dominio abarca las funciones que se describen a continuación.

Entidad de Aplicación. Un problema en aplicaciones distribuidas en red es cómo las aplicaciones pueden contactarse una a la otra. En la red DICOM los socios se reconocen uno a otro por medio de las Entidades de Aplicación. Una Entidad de Aplicación es aquella parte de un proceso que trata (negocia) con la comunicación. Contiene Usuario de Servicio del proceso, incluyendo funciones para establecer conexiones y transferir la información. Una Entidad de Aplicación tiene un nombre (Título de Aplicación) que tiene que usarse cuando se determina la comunicación.

Dirección de Presentación. Los Títulos de Aplicación son nombres simbólicos para los procesos involucrados en la comunicación. En una red real se debe proporcionar una dirección de red. Esta se llama la Dirección de Presentación y apunta a la Entidad de Aplicación mencionada. Se llama dirección de presentación porque el usuario de servicio es la capa de Aplicación (OSI), el proveedor de servicio la capa de Presentación (OSI) y capas inferiores. La frontera entre ambas capas es el punto de acceso a la red donde los datos se transfieren desde la capa de aplicación hasta las capas de red. Cada punto de acceso en la red tiene una dirección única.

El formato de la Dirección de Presentación depende del protocolo de red utilizado. Las redes DICOM, son en la mayoría de los casos, realizadas usando la pila de protocolo TCP/IP. En este caso la dirección de presentación se mapea a un socket TCP/IP.

Negociación de Asociación. La conexión para intercambio de información entre dos Entidades de Aplicación se llama una Asociación. Para una asociación se fijan un número de puntos de comunicación como el contexto en el cual puede darse el intercambio de información. Este contexto, llamado Contexto de Aplicación, se define en el estándar DICOM y ambas partes deben estar de acuerdo sobre la actuación de acuerdo a esta definición de contexto.

Durante el inicio de una asociación, un Identificador Unico (UID) que identifica al Contexto de Aplicación, se transfiere al socio. Por comparación de este UID del Contexto de Aplicación, el socio puede decidir si es capaz de manejar esta petición para una asociación. El socio que inicia la asociación propone las clases

SOP que usará, el papel SCU/SCP para cada clase SOP y la manera de representación de la información.

Después que se procesa esta negociación, ambos socios conocen las capacidades y limitaciones uno de otro. El intercambio de información real puede tomar lugar de acuerdo a la Clase de Servicio y reglas de Clase SOP definidas para estas clases. Cuando una asociación ya no se requiere, se termina.

Contexto de Presentación. Para cada Clase SOP negociada durante la inicialización de una Asociación se tiene que alcanzar un acuerdo entre los procesos involucrados acerca de la sintaxis de transferencia usada entre dichos procesos. El socio que inicia propone todas las sintaxis de transferencia que puede manejar para una cierta Clase SOP. El otro lado selecciona una de estas sintaxis de transferencia, fijando el Contexto de Presentación para esta Clase SOP.

Protocolos de Red. El protocolo de red presente tiene que cumplir con los servicios estándar como están definidos para la pila de protocolo OSI. Para la capa de aplicación tienen que estar disponibles dos grupos de servicios para una implementación DICOM: el protocolo de Control de Asociación (ACSE) y el protocolo de mensajes DICOM (DIMSE). ACSE es un estándar del protocolo OSI, DIMSE implementa los servicios DICOM.

La interfase entre ACSE, DIMSE y la aplicación es la Interfase DICOM como esta descrito en el estándar DICOM. Esta especificación describe cuáles parámetros se requieren para cada función de las peticiones ACSE y DIMSE. ACSE, DIMSE y la interfase DICOM son parte del Contexto de Aplicación DICOM.

Pila de Protocolo TCP/IP. La combinación de una pila TCP/IP y una extensión para servicios de aplicación OSI se usa ampliamente para realizar DICOM a través de red. Debido a que no están definidas capas superiores para TCP/IP, la funcionalidad de la capa de aplicación, presentación y sesión como la requiere DICOM, se describe en su estándar. Esta funcionalidad esta combinada en una capa: la Capa Superior DICOM o DUL (DICOM Upper Layer).

La DUL usa la misma Interfase DICOM tanto para la pila de protocolo TCP/IP como para la pila OSI. En el nivel inferior la DUL tiene una interfase para la capa TCP. La Asociación DICOM entre las entidades de aplicación se mapea a una conexión TCP. La Dirección de Presentación se mapea a un número de puerto TCP, combinado con el número de IP o nombre del servidor. Esta combinación de número de puerto TCP y número de IP se llama la Dirección de Socket.

3.2. Descripción de DIMSE

El modelo orientado a objetos proporciona una manera para describir no solo la información sino que hacer con ella, o como los programas de computadora pueden accesar la información a través de una colección de objetos. En el diseño orientado a objetos, los métodos se asocian con los objetos definidos. DICOM hace uso de este concepto definiendo servicios tales como "almacenar imagen" u "obtener información de paciente". Estos servicios se implementan en DICOM a través de lo que denomina como notificaciones y operaciones. DICOM define un conjunto genérico de notificaciones y operaciones y les da el nombre de Elementos de Servicio de Mensaje DICOM o DIMSE, este nombre corto proviene de *DICOM Message Service Element.*

La parte 7 del estándar DICOM [NEMA37] especifica el DIMSE, definiendo un elemento de Servicio de Aplicación que es usado por un par de Entidades de Aplicación DICOM con el propósito de intercambiar imágenes médicas e información relacionada. Este intercambio de mensajes se lleva a cabo siguiendo un protocolo que define las reglas necesarias de codificación para construir estos mensajes. En términos generales la parte 7 del estándar DICOM especifica lo siguiente:

- Un grupo de primitivas de servicio proporcionado por el Elemento de Servicio de Aplicación DIMSE
- Los parámetros que se pasan en cada primitiva de servicio
- Los procedimientos aplicables a las primitivas de servicio
- La Sintaxis Abstracta del protocolo de comando normalizado y compuesto DICOM y las reglas de codificación asociadas a aplicarse
- Procedimientos para la correcta interpretación del protocolo de control de información
- Los requerimientos de conformidad para ser cumplidos por la implementación de esta parte del Estándar
- El Contexto de Aplicación requerido por las Entidades de Aplicación DICOM
- Los requerimientos de Asociación de las Entidades de Aplicación DICOM
- La Información de Asociación de Aplicación para Entidades de Aplicación DICOM

El Protocolo y Elemento de Servicio de Mensaje DICOM dentro del contexto de la Entidad de Aplicación DICOM usa el modelo de referencia básico OSI para modelar la interconexión de equipo de imágenes médicas. Como se muestra en la figura 3.2 se distinguen siete capas de protocolos de comunicación. DICOM usa el Servicio de Capa Superior OSI para separar el intercambio de Mensajes DICOM, en la Capa de Aplicación, del soporte de comunicación provisto por las capas inferiores.

La frontera de Servicio de Capa Superior OSI permite, a parejas de Entidades de Aplicación, establecer Asociaciones, transferir Mensajes y terminar Asociaciones. Para esta frontera, DICOM ha adoptado los estándares OSI (Servicio de Presentación aumentado por el Elemento de Servicio de Control de Asociación). Es un servicio simple que aísla la Capa de Aplicación DICOM de la pila especifica de protocolos usada en las capas de soporte de comunicación. Se ofrecen tres opciones de comunicación:

1. Una pila mínima OSI de protocolos con un Kernel de Sesión full duplex, Kernel de Presentación, y ACSE (Association Control Service Element). Esto reduce la sobrecarga de capa superior mientras mantiene conformidad completa para los Estándares de protocolo OSI

2. Un protocolo de Capa Superior aumentado TCP/IP. Combina los protocolos de capa superior OSI en un protocolo sencillo simple de implementar mientras provee los mismos servicios y funciones ofrecidos por la pila OSI

3. una pila de protocolo punto a punto compatible con las versiones previas del Estándar

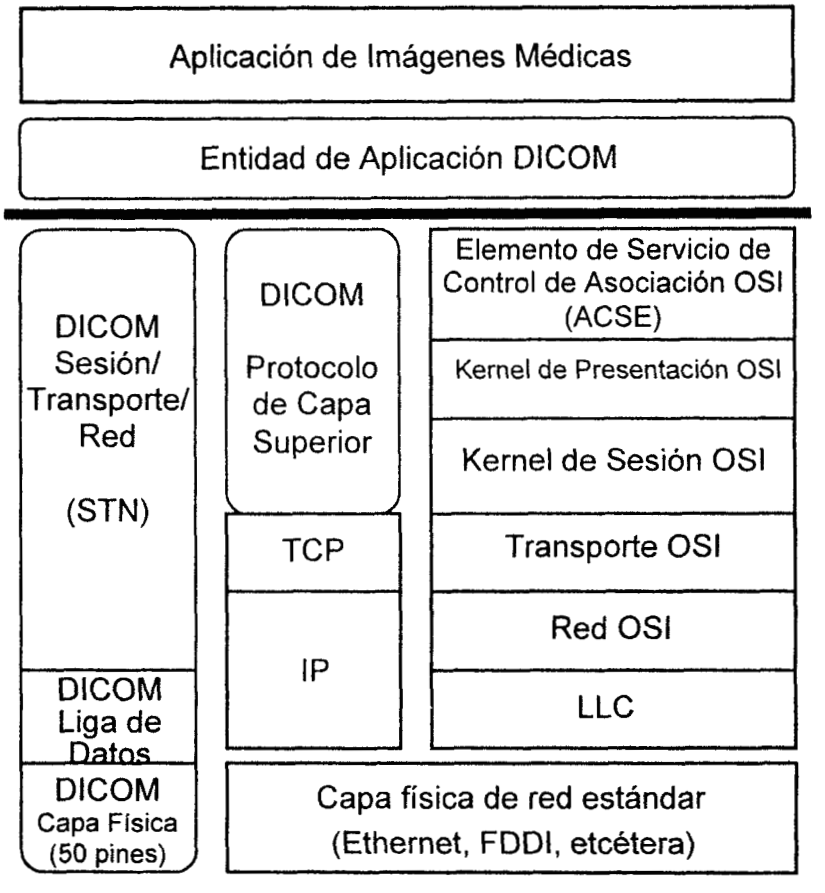

Figura 3.2 DICOM y el modelo de referencia básico OSI

La Entidad de Aplicación DICOM usa los servicios de datos de Asociación y Presentación del Servicio de Capa Superior OSI definido en la Parte 8 del estándar [NEMA38]. El Elemento de Servicio de Control de Asociación (ACSE) aumenta el Servicio de Capa de Presentación con establecimiento de Asociación y servicios de terminación. En el caso de TCP/IP, el equivalente completo de ACSE es provisto por el Servicio de Capa Superior DICOM. Para la pila punto a punto de DICOM, un subconjunto mínimo de ACSE se proporciona por el Servicio Sesión/Transporte/Red. Una Entidad de Aplicación DICOM y los Elementos de Servicio incluidos se muestran en la Figura 3.3.

La Entidad de Aplicación DICOM usa los servicios provistos por el Elemento de Servicio de Mensaje DICOM. El Elemento de Servicio de Mensaje DICOM especifica dos grupos de servicios:

- DIMSE-C soporta operaciones asociadas con Clases SOP compuestas y provee compatibilidad efectiva con las versiones previas del Estándar DICOM.

- DIMSE-N soporta operaciones asociadas con Clases SOP normalizadas y provee un conjunto extendido de operaciones y notificaciones orientadas a objetos. Se basa en el Modelo de Administración del Sistema OSI y específicamente sobre la definición de Servicios de Información de Administración Común OSI (CMIS).

3.2.1. Estructura de mensaje DICOM y grupo de comandos.

La información se comunica a través de la interfase de red DICOM en un mensaje DICOM. Un mensaje esta compuesto de un Grupo de Comandos seguido por un Grupo de datos condicional. El Grupo de Comandos se usa para indicar las operaciones/notificaciones que se ejecutan sobre o con el Grupo de Datos.

Un Grupo de Comandos esta construido de Elementos de Comando. Los Elementos de Comando contienen los valores codificados para cada campo individual del Grupo de Comandos por la semántica especificada en el protocolo DIMSE. Cada Elemento de Comando esta compuesto de una Etiqueta explícita, una Longitud de Valor, y un Campo de Valor. La estructura completa de un Mensaje DICOM se muestra en la Figura 3.4.

Los Elementos de Comando en un Grupo de Comando estarán ordenados por incremento del número de Etiqueta del Elemento de Comando. Una Etiqueta del Elemento de Comando únicamente identifica un Elemento de Comando y ocurrirá muchas veces en un Grupo de Comando. La codificación del Grupo de Comandos será con Ordenación de Byte Little Endian (primero el byte menos significativo, los bytes más a la derecha son los más significativos).

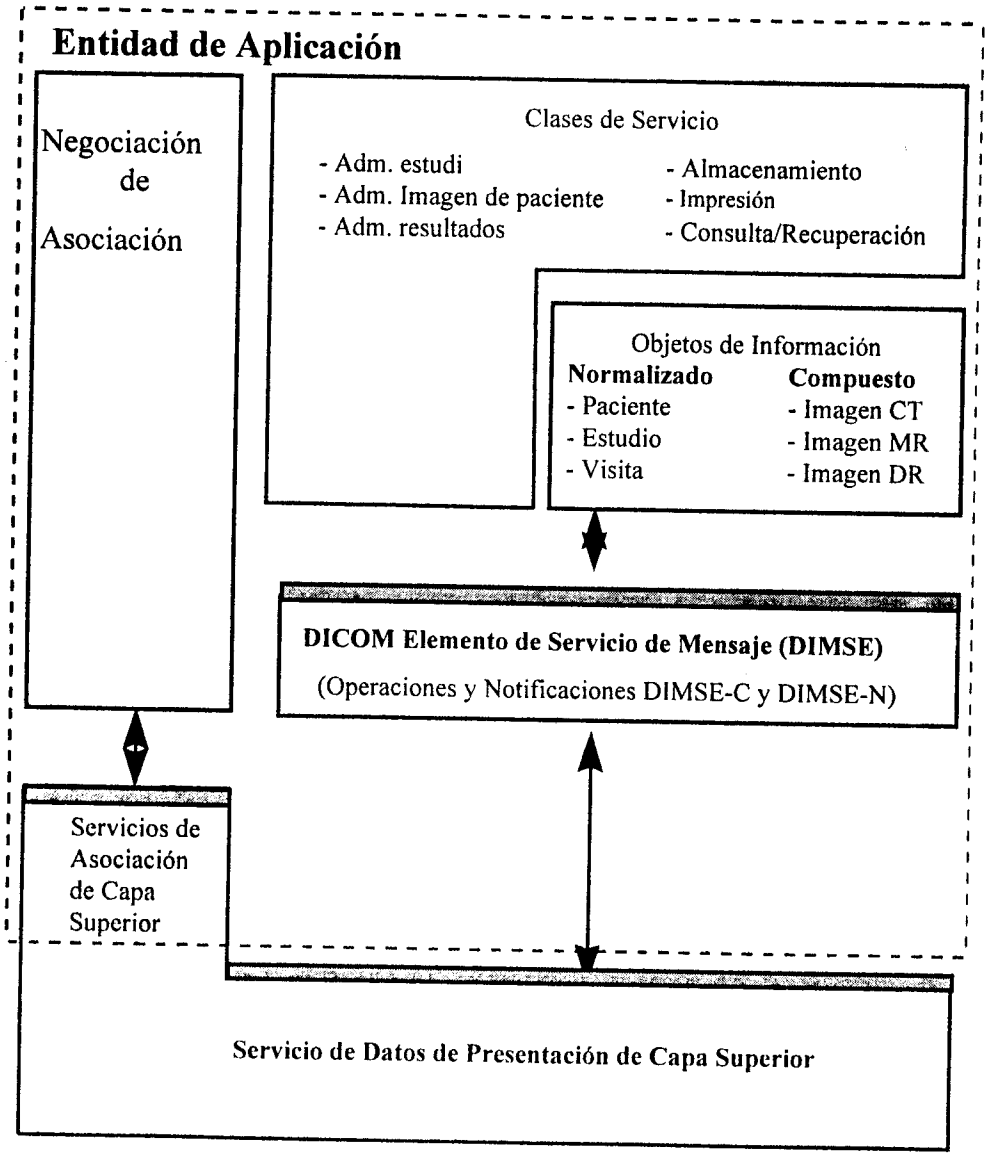

Figura 3.3 Estructura de capa de aplicación DICOM

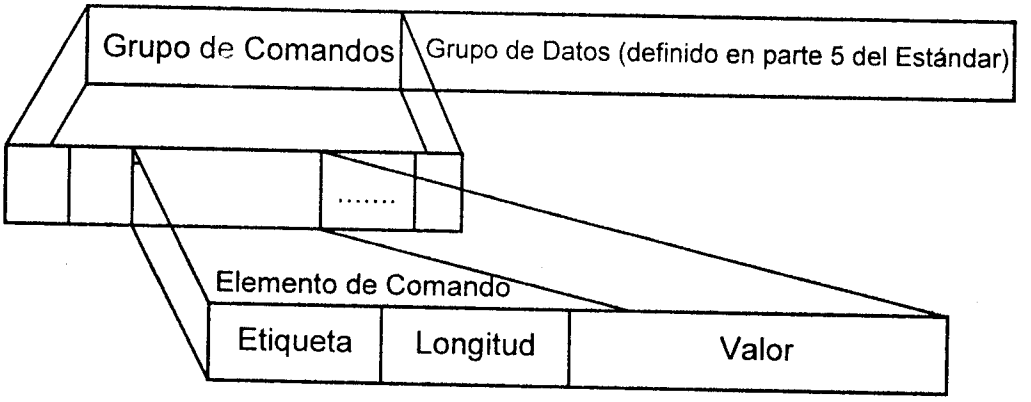

Figura 3.4 Estructura de Mensaje DICOM

Como ya se mencionó, un Elemento de Comando esta compuesto de tres campos:

1. Etiqueta de Elemento de Comando: Un par ordenado de enteros sin signo de 16 bits representando el Número de Grupo seguido por el Número de Elemento.

2. Longitud de Valor: Un entero sin signo de 32 bits representando la longitud explícita como el número par de bytes que constituye el Valor. No incluye los campos de la longitud de la Etiqueta del Elemento de Comando o Longitud de Valor.

3. Campo de Valor: Un número par de bytes conteniendo el Valor(es) del Elemento de Comando. El tipo de Valor(es) almacenado en este campo esta especificado por la Representación de Valor (VR) del Elemento de Comando. El VR para un Elemento de Comando dado puede determinarse usando el Diccionario de Comandos en el Anexo E de la parte 7 del Estándar.

3.2.2. Descripción del servicio

El Elemento de Servicio de Mensaje DICOM soporta la comunicación entre parejas de usuarios de servicio DIMSE. Un usuario de servicio DIMSE actúa en uno de dos roles:

a) invocando un servicio de usuario DIMSE
b) ejecutando un servicio de usuario DIMSE

Los usuarios de servicio DIMSE hacen uso de primitivas de servicio proporcionadas por el proveedor de servicio DIMSE. El proveedor de servicio DIMSE es una abstracción de la totalidad de aquellas entidades que proveen servicios DIMSE a parejas de usuarios de servicio DIMSE. Una primitiva de servicio debe ser de los siguientes tipos:

a) primitiva de petición
b) primitiva de indicación
c) primitiva de respuesta
d) primitiva de confirmación

Estas primitivas se muestran en la figura 3.5 y se usan de la siguiente manera para completar exitosamente un servicio DIMSE:

- El usuario de servicios DIMSE que invoca (invocador) emite una primitiva de petición de cualquiera de los servicios DIMSE para el proveedor de servicios DIMSE (proveedor).

- El proveedor recibe la primitiva de petición desde el invocador y emite una primitiva de indicación al usuario de servicios DIMSE que ejecuta (ejecutor).
- El ejecutor recibe la primitiva de indicación del proveedor y ejecuta el servicio solicitado.
- El ejecutor emite una primitiva de respuesta al proveedor.
- El proveedor recibe la primitiva de respuesta del ejecutor y emite una primitiva de confirmación al usuario invocador.
- El invocador recibe la primitiva de confirmación del proveedor completándose el servicio DIMSE.

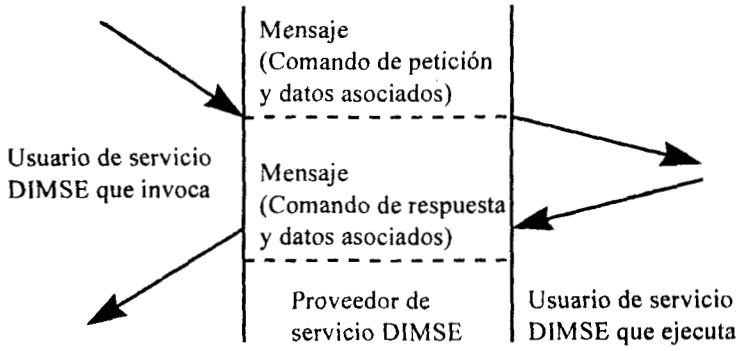

Figura 3.5 Primitivas de servicio DIMSE

DIMSE provee dos tipos de servicios de transferencia de información:

a) Servicio de notificación
b) Servicio de operación

Los servicios de notificación habilitan a una Entidad de Aplicación DICOM para notificar acerca de la ocurrencia de un evento o cambio de estado. La definición de la notificación y el funcionamiento consecuente de las Entidades de Aplicación es dependiente de la Clase de Servicio y Definiciones de Objeto de Información. Los servicios de operación habilitan una Entidad de Aplicación DICOM para solicitar explícitamente una operación para ser ejecutada sobre una instancia SOP administrada por otra Entidad de Aplicación DICOM.

DIMSE recibe solicitudes de operación y notificación y su información relacionada desde el usuario de servicio DIMSE. Una notificación u operación se implementa como una interacción petición/respuesta realizada dentro del contexto de una Asociación establecida. Típicamente, un usuario de servicio DIMSE solicita que una operación particular sea ejecutada (o la notificación sea procesada) y el otro usuario de servicio DIMSE intenta ejecutar la operación (o procesa la notificación) y después reporta el resultado del intento.

Cuando toma parte en las operaciones o notificaciones, el usuario de servicio DIMSE asume uno de los siguientes dos papeles:

a) Ejecuta operaciones las cuales fueron invocadas por su pareja de usuario de servicio DIMSE. Puede también emitir notificaciones de cambio de estado para instancias SOP para uno o más usuarios de servicio DIMSE. Estas notificaciones pueden ser invocadas como un resultado de operaciones iniciadas por otros usuarios de servicio DIMSE.

b) Invoca la ejecución de una operación sobre su pareja de usuario de servicio DIMSE. Puede también recibir notificaciones desde su pareja de usuario de servicio DIMSE.

Estos roles se dibujan en la figura 3.6.

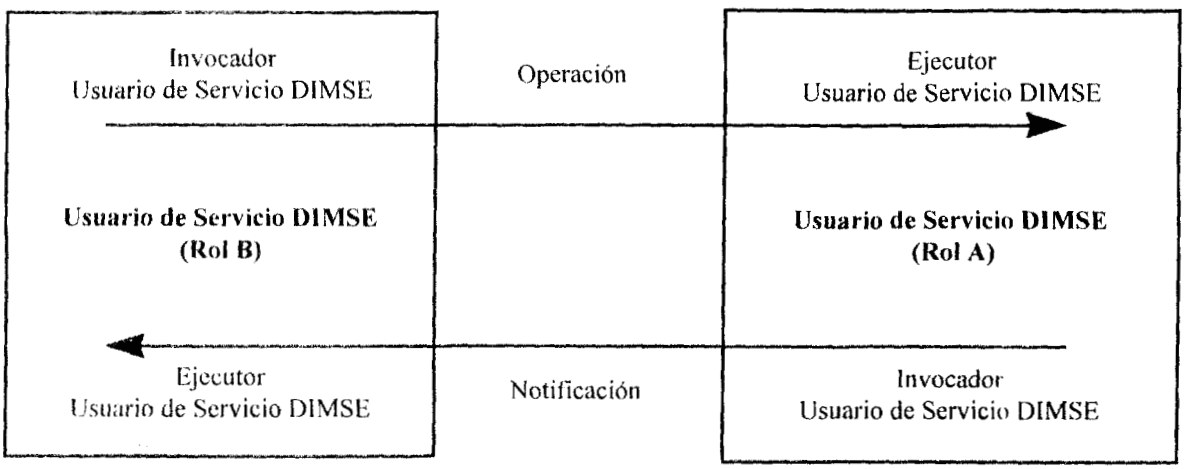

Figura 3.6 Flujo de operación y notificación

Las operaciones y notificaciones, sobre una Asociación establecida, se usan en uno de los siguientes dos modos:

a) Síncrono
b) Asíncrono

En el modo síncrono, el usuario de servicio DIMSE que invoca, requiere una respuesta desde el usuario de servicio DIMSE que ejecuta antes de invocar otra operación o notificación.

En el modo asincrono, el usuario de servicio DIMSE que invoca, puede continuar invocando operaciones o notificaciones adicionales sobre el ejecutor sin esperar una respuesta.

La selección de modo (síncrono o asíncrono) se determina en el tiempo del establecimiento de Asociación. El modo síncrono sirve como el modo de omisión y deberá estar soportado por todos los usuarios de servicio DIMSE. El modo asíncrono es opcional y el número máximo de operaciones/negociaciones sin resolver se negocia durante el establecimiento de Asociación.

Durante la fase de establecimiento de Asociación, figura 3.7, un usuario de servicio DIMSE intercambiará información de inicialización usando parámetros del Servicio de Capa Superior A-ASSOCIATE el cual incluye:

- Contexto de aplicación
- Requerimientos de presentación y sesión
- Información de usuario específico DIMSE
- Información de Asociación de Aplicación

El servicio A-ASSOCIATE, definido en la Parte 8, es invocado por un usuario de servicio DIMSE para establecer una Asociación con una pareja de usuario de servicio DIMSE. El establecimiento de Asociación es siempre la primera fase del Intercambio de Mensaje DICOM.

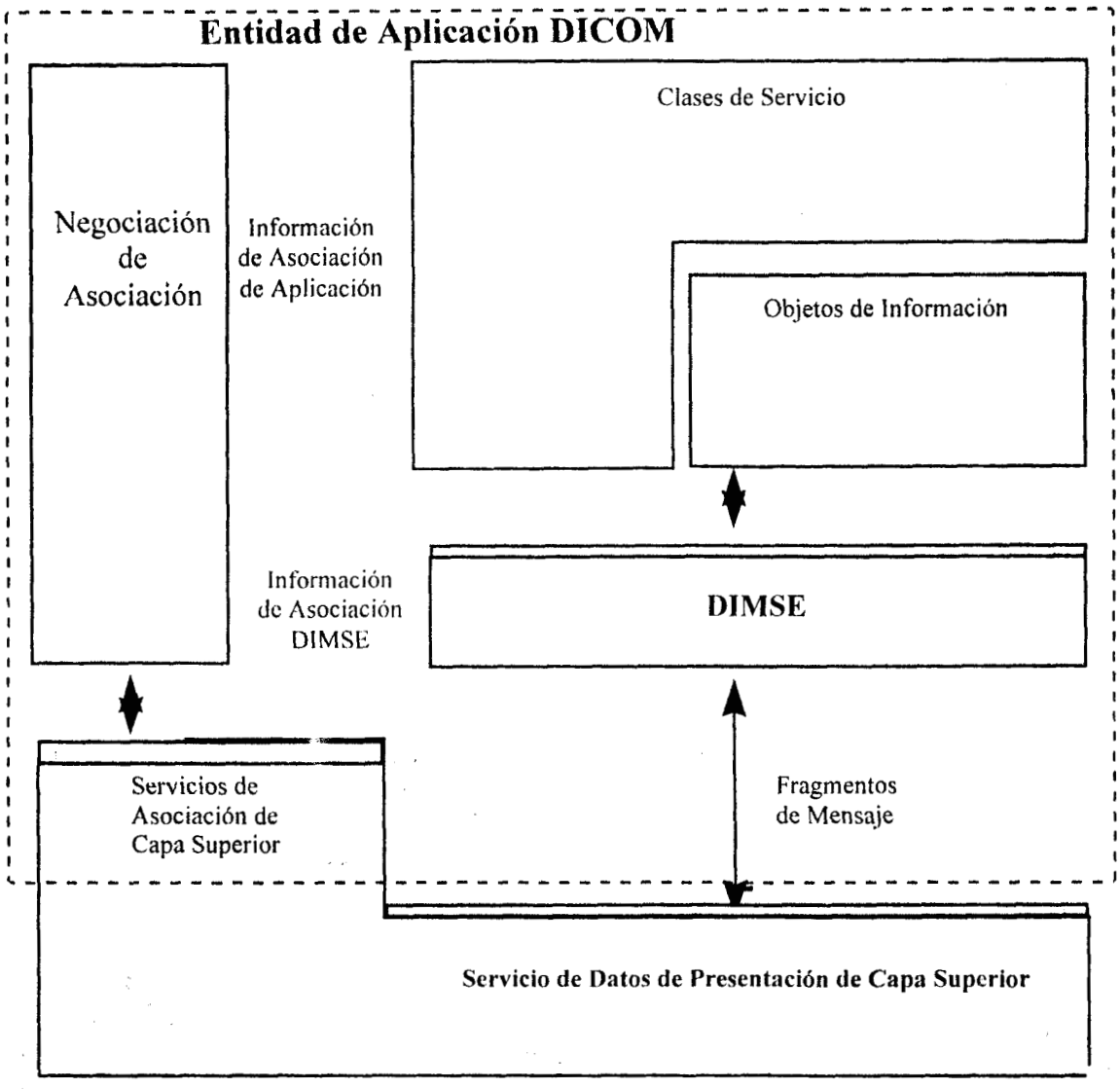

Figura 3.7 Entidad de Aplicación DICOM y Asociación

El usuario de servicio DIMSE que invoca y el usuario de servicio DIMSE que responde incluirán Información de Asociación de Aplicación en la primitiva de petición y respuesta respectivamente. El significado de este parámetro es específico al Contexto de Aplicación.

El Servicio A-RELEASE, definido en la Parte 8, se invoca por un usuario de servicio DIMSE para solicitar la terminación ordenada de una Asociación entre una pareja de usuarios de servicio DIMSE.

El Servicio A-ABORT, definido en la Parte 8, se invoca por un usuario de servicio DIMSE para solicitar la terminación abrupta de la Asociación entre una pareja de usuarios de servicio DIMSE. El A-ABORT del invocador deberá incluir (dentro del campo de información de usuario A-ABORT) el parámetro Fuente de Aborto. El parámetro Fuente de Aborto indica la fuente que inicia el aborto. Toma uno de los siguientes valores simbólicos:

- Proveedor de servicio DIMSE
- Usuario de servicio DIMSE

3.2.3. Servicios DIMSE

Debido a la manera en la cual se aplican las operaciones a Instancias SOP, están definidos dos grupos de servicios DIMSE:

1. DIMSE-N: aquellos servicios aplicables a Instancias SOP Normalizadas
2. DIMSE-C: aquellos servicios aplicables a Instancias SOP Compuestas

Tabla 1. Servicios DIMSE

Nombre	Grupo	Tipo
C-STORE	DIMSE-C	operación
C-GET	DIMSE-C	operación
C-MOVE	DIMSE-C	operación
C-FIND	DIMSE-C	operación
C-ECHO	DIMSE-C	operación
N-EVENT-REPORT	DIMSE-N	notificación
N-GET	DIMSE-N	operación
N-SET	DIMSE-N	operación
N-ACTION	DIMSE-N	operación
N-CREATE	DIMSE-N	operación
N-DELETE	DIMSE-N	operación

Los Servicios DIMSE-C permiten a una Entidad de Aplicación DICOM solicitar explícitamente una operación a otra Entidad de Aplicación DICOM sobre Instancias SOP Compuestas. Las operaciones permitidas están propuestas para

ser efectivamente compatibles con aquellas provistas por versiones previas de este Estándar. DIMSE-C provee solamente servicios de operación.

DIMSE-C provee los siguientes servicios de operación, todos los cuales son servicios confirmados y como tales, se espera una respuesta:

1. El servicio C-STORE es invocado por un usuario de servicio DIMSE para solicitar el almacenamiento de una Instancia SOP Compuesta de información en otro usuario de servicio DIMSE semejante.

2. El servicio C-FIND es invocado por un usuario de servicio DIMSE para comparar un conjunto de Atributos contra los atributos del conjunto de Instancias SOP administradas por un usuario de servicio DIMSE semejante. Regresa una lista de Atributos con los valores de aquellos atributos que fueron igualados.

3. El servicio C-GET es usado por un usuario de servicio DIMSE para comparar un conjunto de Atributos contra los Atributos de un conjunto de Instancias SOP compuestas mantenidos por un usuario de servicio DIMSE semejante, y recupera todas las Instancias SOP compuestas que son igualadas. Dispara una o más suboperaciones sobre la misma asociación.

4. El servicio C-MOVE es invocado por un usuario de servicio DIMSE para mover la información de una o más Instancias SOP Compuestas desde un usuario de servicio DIMSE semejante, a un tercer usuario de servicio DIMSE, basado sobre los atributos suministrados por el usuario de servicio DIMSE invocador. Dispara una o más suboperaciones C-STORE en una asociación separada.

5. El servicio C-ECHO es invocado por un usuario de servicio DIMSE para verificar comunicaciones punto a punto con un usuario de servicio DIMSE semejante.

Los servicios DIMSE-N proveen tanto servicios de notificación y operación aplicables a Instancias SOP Normalizadas.

1. DIMSE-N provee un solo Servicio de Notificación, el N-EVENT-REPORT. El servicio N-EVENT-REPORT es invocado por un usuario de servicio DIMSE para reportar un evento acerca de una Instancia SOP a un usuario de servicio DIMSE semejante.

2. El servicio N-GET es invocado por un usuario de servicio DIMSE para solicitar la recuperación de información desde un usuario de servicio DIMSE semejante.

3. El servicio N-SET es invocado por un usuario de servicio DIMSE para solicitar la modificación de información por un usuario de servicio DIMSE semejante.

4. El servicio N-ACTION es invocado por un usuario de servicio DIMSE para solicitar a un usuario de servicio DIMSE semejante que ejecute una acción.

5. El servicio N-CREATE es usado por un usuario de servicio DIMSE para pedir a un usuario semejante crear una nueva Instancia SOP administrada, completa con su identificación y los valores de sus Atributos asociados y simultáneamente registrar su identificación.

6. El servicio N-DELETE es usado por un usuario de servicio DIMSE para pedir a un usuario semejante borrar una Instancia SOP manejada y borrar el registro de su identificación.

Algunos servicios DIMSE son atómicos en el sentido de que el servicio se ejecuta por una operación o notificación. En tal caso las primitivas de servicio DIMSE se usan por pares de usuarios de servicio DIMSE para invocar y ejecutar la operación o notificación.

Otros servicios DIMSE requieren el uso de una o más suboperaciones para ejecutar el servicio. En tales casos las primitivas de servicio DIMSE se usan por un par de usuarios de servicio DIMSE para invocar y ejecutar cada suboperación. Cómo y cuándo las primitivas de servicio de suboperación se usan esta definido por los procedimientos para el servicio DIMSE.

Cada servicio DIMSE requiere una o más primitivas de respuesta como un resultado de la invocación del servicio. Cómo y cuándo las primitivas de respuesta múltiple se usan esta definido por los procedimientos para el servicio DIMSE. El regreso de respuestas múltiples esta condicionado sobre la información incluida en la primitiva de petición por el usuario de servicio DIMSE.

Ciertos servicios DIMSE permiten la cancelación del servicio por el uso de primitivas. Esto permite a un usuario de servicio DIMSE invocador solicitar la terminación de un servicio DIMSE después de completar la primitiva de servicio solicitada pero antes de completar la primitiva de servicio de confirmación.

3.2.4. Descripción del protocolo DIMSE

La máquina de protocolo DIMSE define los procedimientos y las reglas de codificación necesarias para construir Mensajes usados para intercambiar solicitudes y respuestas de comando entre pares de usuarios de servicio DIMSE (por ejemplo, dos Entidades de Aplicación DICOM). La relación entre Mensajes y los diferentes tipos de primitivas de servicio se muestran en la figura 3.5.

La máquina de protocolo DIMSE acepta primitivas de servicio de petición y respuesta de usuario de servicio DIMSE y construye Mensajes definido por los procedimientos definidos para los diferentes servicios. La máquina de protocolo

DIMSE acepta Mensajes y los pasa al usuario de servicio DIMSE por medio de primitivas de servicio de indicación y confirmación.

Los procedimientos definen las reglas para la transferencia de mensajes que transportan los comandos de petición y respuesta. Estas reglas definen la interpretación de los varios campos en la parte del Mensaje del comando; no definen que hará el usuario de servicio DIMSE invocador con la información (la parte del Conjunto de Datos del Mensaje) que pidió ni como un usuario de servicio DIMSE ejecutor debiera procesar la información.

La primitiva de petición del usuario de servicio DIMSE resulta en un Mensaje llevando un Comando de Petición (con un Conjunto de Datos Asociado). Cada Mensaje induce una primitiva de indicación para el usuario de servicio DIMSE ejecutor.

Las primitivas de respuesta del usuario de servicio DIMSE ejecutor resultan en un Mensaje llevando un Comando de Respuesta (con un Conjunto de Datos asociado opcional). Cada Mensaje induce una primitiva de confirmación para el usuario de servicio DIMSE que invoca.

El protocolo necesario para ejecutar el conjunto de operaciones DIMSE-C es similar para cada una de ellas, y solo se presentan pequeñas variaciones en algunas de las operaciones. Como ejemplo representativo del protocolo que se sigue en las operaciones compuestas, describiremos el protocolo del servicio C-STORE.

La información necesaria para las primitivas DIMSE-C de petición e indicación C-STORE son transportadas en el Mensaje C-STORE-RQ. La información necesaria para las primitivas DIMSE-C de respuesta y confirmación C-STORE se transportan en el mensaje C-STORE-RSP.

Los procedimientos del protocolo C-STORE son iniciados por el invocador del usuario de servicio DIMSE emitiendo una primitiva de petición C-STORE. En la recepción de la primitiva de petición C-STORE la máquina de protocolo DIMSE-C debe:

- Construir un mensaje llevando el C-STORE-RQ
- Enviar el mensaje usando el servicio de petición P-DATA

En la recepción de mensaje llevando un C-STORE-RQ la maquina de protocolo DIMSE-C emitirá una primitiva de indicación C-STORE al ejecutor.

En la recepción de la primitiva de respuesta C-STORE, emitida por el ejecutor, la máquina de protocolo DIMSE-C:

- Construirá un mensaje llevando el C-STORE-RSP

- Enviará el mensaje usando el servicio de petición P-DATA

En la recepción de un mensaje llevando un C-STORE la máquina de protocolo emitirá una primitiva de confirmación C-STORE al invocador, completando así el procedimiento C-STORE.

El protocolo para los servicios DIMSE-N es igual para todos ellos, como ejemplo, se muestra el correspondiente al servicio N-GET. La información necesaria para las primitivas de petición e indicación de N-GET se lleva en el Mensaje N-GET-RQ. La de respuesta y confirmación se lleva en el Mensaje N-GET-RSP.

Los procedimientos del protocolo N-GET son iniciados por el invocador emitiendo una primitiva de petición N-GET. En la recepción de esta primitiva la máquina de protocolo:

- Construirá un mensaje llevando el N-GET-RQ.

- Enviará el mensaje usando el servicio de petición P-DATA.

En la recepción de un Mensaje que lleva un N-GET-RQ la máquina de protocolo emitirá una primitiva de indicación N-GET al ejecutor.

En la recepción de la primitiva de respuesta, emitida por el ejecutor, la máquina de protocolo:

- Construirá en un mensaje llevando el N-GET-RSP.

- Enviará el mensaje usando el servicio de petición P-DATA.

En la recepción de un mensaje N-GET-RSP la máquina emitirá una primitiva de confirmación N-GET al invocador, completando así el procedimiento.

3.3. Herramientas de la tecnología de objetos

3.3.1. Proceso de desarrollo

La finalidad del presente documento es describir el proceso que se aplicará en el proyecto de software DIMSE, que tiene como objetivo resolver el problema de analizar, diseñar y desarrollar un modelo de solución para que un par de Entidades de Aplicación DICOM puedan intercambiar mensajes. Una de las premisas básicas en este proyecto es aplicar el modelo de objetos en cada una de las fases del mismo, para satisfacer esta premisa y aplicar una metodología que permita que el proyecto sea sano y con probabilidades de éxito, a continuación se describe el proceso de desarrollo sugerido por Craig Larman [Larman98], el cual está basado en años de experiencia en el arte de análisis y diseño orientado a objetos, además de utilizar el Lenguaje de Modelado Unificado (UML), el cual ha surgido como un estándar de notación para el modelado de sistemas usando conceptos de orientado a objetos.

Un proceso de desarrollo de software es un método para organizar las actividades para la creación, desarrollo y mantenimiento de sistemas de programas para computadora. Se recomiendan los siguientes principios como parte del desarrollo [Booch96]:

- Iterativo e incremental. Un ciclo de vida iterativo esta basado en la ampliación y refinamiento sucesivo de un sistema a través de múltiples ciclos de desarrollo de análisis, diseño, implementación y prueba. El sistema crece agregando funciones nuevas con cada ciclo de desarrollo.
- Manejado por casos de uso. El proceso de desarrollo deberá estar influenciado y organizado alrededor de los casos de uso. Por ejemplo, los requerimientos se organizan y expresan en casos de uso, las actividades dentro de un ciclo de desarrollo se enfocan en la satisfacción de los casos de uso, los conceptos y clases de software se pueden identificar de las consideraciones de los casos de uso.
- Enfasis temprano en la definición de la arquitectura. Arquitectura se refiere a la estructura de alto nivel de subsistemas y componentes, y sus interfases, conexiones, e interacciones. Consiste de un conjunto de marcos de trabajo, subsistemas, clases, asignación de responsabilidades y colaboración de objetos que satisfacen las funciones del sistema.

Fundamentalmente, la descripción de un proceso incluye las actividades desde los requerimientos hasta la entrega. Adicionalmente, un proceso completo abarca temas amplios relacionados a la industrialización del desarrollo de software, tal como ciclo de vida de largo plazo de un producto, documentación, soporte y entrenamiento, trabajo paralelo y coordinación entre partes. El proceso que se describirá aquí solo contempla las actividades básicas del mismo y no contempla

los pasos relativos a la industrialización del software porque [Larman98] menciona que está fuera del alcance de su libro.

La figura 3.8 muestra una descripción general del proceso, así como el orden en el cual deben de recorrerse, las diferentes fases en este proceso de desarrollo de proyectos de software.

Craig Larman establece que a un alto nivel, los pasos principales para liberar una aplicación incluyen lo siguiente:

1. Planear y Elaborar. Planeación, definición de requerimientos, construcción de prototipos y así sucesivamente.
2. Construir. La construcción del sistema
3. Difundir. El uso de la implementación.

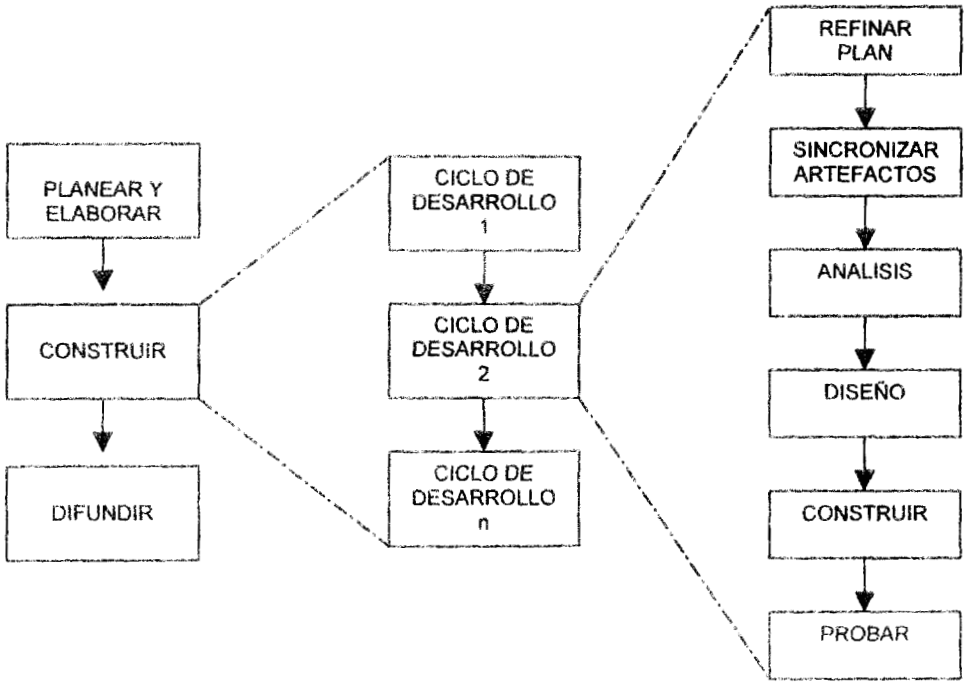

Figura 3.8 Pasos principales y ciclos de desarrollo del proceso

Un ciclo de vida iterativo esta basado en agrandar y refinar en forma sucesiva un sistema a través de ciclos de desarrollo múltiples de análisis, diseño, implementación, y prueba.

El sistema crece agregando nuevas funciones con cada ciclo de desarrollo. Después de una fase preliminar de Planear y Elaborar, sigue el desarrollo en una fase de Construir a través de varios ciclos de desarrollo.

Cada ciclo enfrenta un conjunto relativamente pequeño de requerimientos, avanzando a través de las siguientes etapas:

- Refinar el plan
- Sincronizar artefactos
- Análisis
- Diseño
- Construcción
- Prueba

3.3.1.1. Planear y Elaborar

La fase de Planear y Elaborar de un proyecto incluye la concepción inicial, alternativas de investigación, planeación, especificación de requerimientos, etc.

Ejemplo de una lista de actividades tipo de esta fase es la siguiente:

1. Definir un plan
2. Crear un reporte de la investigación preliminar
3. Definir los requerimientos
4. Registrar términos en un glosario
5. Implementar prototipo
6. Definir los casos de uso
7. Definir el modelo conceptual
8. Definir la arquitectura del sistema
9. Refinar el plan

Las cosas que se pueden generar en esta fase pueden incluir las siguientes:

- Plan: agenda, recursos, presupuesto, etc.
- Reporte preliminar: motivación, alternativas, necesidades de negocio.
- Especificación de requerimientos: declaración enunciativa de requerimientos.
- Glosario: diccionario de términos y cualquier información asociada, tal como restricciones y reglas.
- Prototipo: sistema prototipo creado para ayudar a entender el problema sus riesgos y requerimientos.
- Casos de uso: descripción de los procesos del dominio.
- Diagramas de casos de uso: ilustración de todos los casos de usos y sus relaciones.
- Modelo conceptual: modelo conceptual preliminar que ayude a entender el vocabulario del dominio, especialmente en su relación a los casos de uso y especificación de requerimientos.

Con el fin de aclarar las actividades de esta fase, a continuación se explican con más detalle algunos conceptos que son de importancia en esta etapa del desarrollo.

3.3.1.1.1. Requerimientos

Los requerimientos son la descripción de las necesidades o deseos de un producto. La meta principal de los requerimientos es identificar y documentar lo que realmente es necesario, en una forma que comunique claramente al cliente y al equipo de desarrolladores. El reto es definir los requerimientos sin ambigüedad, tal que se identifiquen los riesgos para que cuando se entregue el producto no haya sorpresas. La siguiente muestra de artefactos se recomiendan en esta parte:

- Declaración descriptiva
- Clientes
- Metas
- Funciones del sistema
- Atributos del sistema

Las funciones de un sistema son lo que se supone que debe hacer el sistema. Para verificar que **algo** es en verdad una función del sistema, ese algo debe tener sentido en la siguiente oración:

*El sistema deberá hacer (**algo**)*

En contraste, los atributos del sistema son cualidades no funcionales del sistema, tal como facilidad de uso, con frecuencia los atributos se confunden con funciones. Sin embargo, debe tenerse claro que los atributos son características o dimensiones del sistema.

3.3.1.1.2. Casos de Uso

Los casos de uso son una herramienta para mejorar el entendimiento de los requerimientos. Un caso de uso es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que usa un sistema para completar un proceso [Jacobson92].

Un método para identificar casos de uso basado en actores es:

- Identificar los actores relacionados con un sistema u organización.
- Para cada actor, identificar los procesos que ellos inician o los que participan.

Un segundo método para identificar casos de uso esta basado en eventos:

- Identificar los eventos externos a los que un sistema debe responder.
- Relacionar los eventos con actores y casos de uso.

Los casos de uso se pueden expresar en dos formatos: de alto nivel y extendido. El primero describe un proceso muy brevemente, generalmente con dos o tres oraciones. El segundo describe un proceso con más detalle, se diferencia del primero en que tiene una sección de Curso Típico de Eventos, la cual describe los eventos paso a paso.

Un error común es identificar casos de uso para representar pasos individuales, operaciones o transacciones como casos de uso. Un caso de uso es una descripción de un proceso relativamente grande de principio a fin que incluye muchos pasos o transacciones, normalmente no es un paso o actividad dentro de un proceso.

3.3.1.2. Construir

Una vez que se ha completado la fase de Planear y Elaborar, y que se han identificado los casos de uso, se empieza la fase de Construir, dentro de la cual ocurren los ciclos de desarrollo iterativo.

La fase de construcción de un proyecto involucra el repetir ciclos de desarrollo en los cuales se extiende el sistema. Cada ciclo de desarrollo esta compuesto por los pasos que se muestran en la figura 3.9.

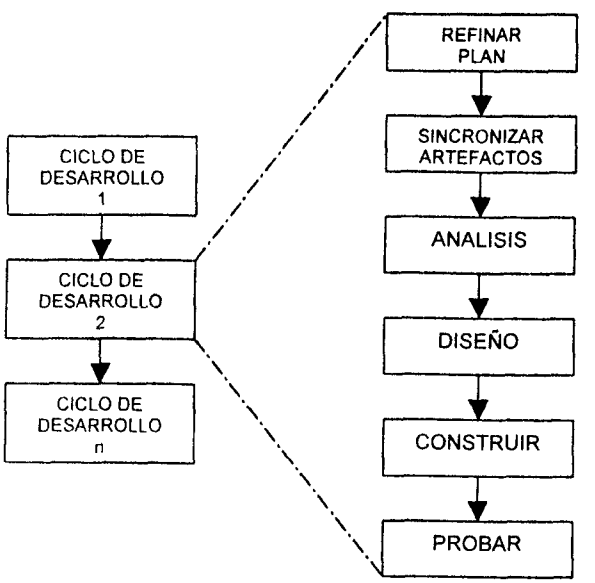

Figura 3.9 Fase de Construir y sus ciclos de desarrollo

Las actividades iniciales dentro de esta fase están relacionadas con la administración del proyecto. En el caso general, a esto le sigue (lo más probable es que ocurra en paralelo) una sincronización de documentación (por ejemplo, diagramas) con el estado actual del proyecto, y después se entra a una fase de análisis, dentro de la cual se investigan de cerca los problemas del ciclo actual.

Dentro de un solo ciclo de desarrollo, los pasos más importantes son el análisis y el diseño. Las actividades en cada uno de estos pasos se muestran en la figura 3.10.

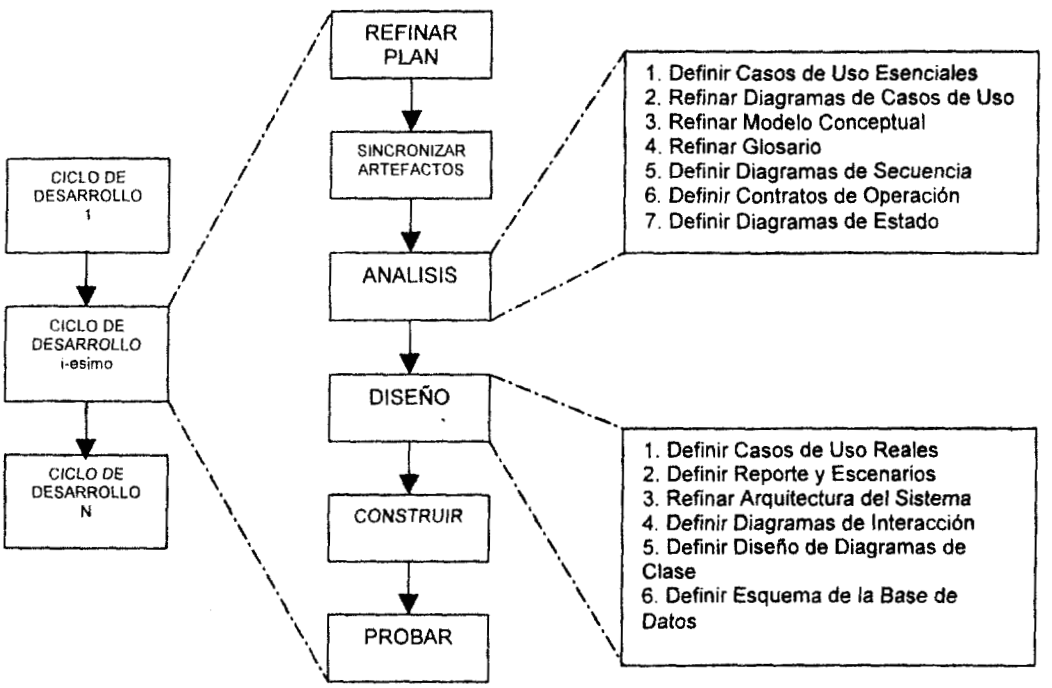

Figura 3.10 Actividades de los pasos de Análisis y de Diseño.

Para crear una aplicación, se requiere una descripción del problema y los requerimientos, cuál es el problema en cuestión y qué debe hacer el sistema. El análisis enfatiza la investigación del problema en lugar de definir una solución, es decir, el análisis investiga lo que debe hacer el sistema y no se preocupa del como debe hacerlo. En el análisis existe un énfasis en encontrar y describir los objetos o conceptos en el dominio del problema.

Para crear la aplicación, también se necesitan descripciones detalladas de la solución lógica y como satisfacer los requerimientos. El diseño enfatiza la solución lógica, como se satisfacen los requerimientos, esta parte del desarrollo tiene que ver con el cómo satisfacer las necesidades del sistema de software.

3.3.1.2.1. Análisis

Dentro de la fase de análisis, una de las primeras actividades es el desarrollo de un modelo conceptual.

3.3.1.2.1.1. Modelo Conceptual

La identificación de conceptos es parte de una investigación del dominio del problema. Una cualidad crítica de un modelo conceptual es que es una representación de cosas del mundo real, no de componentes de software.

El paso esencial orientado a objetos en el análisis es la descomposición del problema en conceptos individuales u objetos, las cosas de las cuales estamos enterados. Un modelo conceptual es una representación de conceptos en el dominio del problema. En UML, un modelo conceptual se ilustra con un conjunto de diagramas de estructura estática en las cuales no se definen operaciones. El término modelo conceptual tiene la ventaja de enfatizar el enfoque sobre los conceptos del dominio, no en las entidades de software.

La creación del modelo conceptual empieza haciendo una selección de conceptos candidatos de la siguiente lista, la cual contiene muchas categorías comunes que son dignas de ser consideradas, sin un orden particular de importancia.

Categoría de Concepto	Ejemplo
Objetos físicos o tangibles	Tomógrafo, Cama
Especificaciones, diseños, o descripciones de cosas	Especificación de producto Descripción de mantenimiento
Lugares	Terapia intensiva, Quirófano
Transacciones	Admisión, Egreso
Papeles de la gente	Médico, Enfermera
Contenedores de otras cosas	Hospital, Almacén
Cosas en un contenedor	Pacientes, Medicamentos
Otra computadora o sistema electrónico externo a nuestro sistema	Sistema de autorización de tarjetas de crédito, Control de aduana
Conceptos de nombres abstractos	Salud, Síndrome
Organizaciones	Medicina preventiva, Trabajo social
Eventos	Cirugía, Parto
Procesos (frecuentemente no representados como conceptos, pero puede ser)	Valorar paciente, Diagnóstico clínico
Reglas y políticas	Política de admisión, Política de altas
Catálogos	Catálogo de medicamentos, Catálogo de personal
Manuales, libros	Manual del empleado, Manual de reparación

Otra técnica útil es identificar el sustantivo o frase sustantiva en las descripciones textuales del dominio del problema, y considerarlas como conceptos candidatos o atributos.

Aplique los siguientes pasos para crear un modelo conceptual:

1) Liste los conceptos candidatos usando la lista de Categorías de Conceptos e identificación de frase sustantiva relacionados con los requerimientos.
2) Dibújelos en un modelo conceptual.
3) Agregue las asociaciones necesarias para registrar relaciones de las cuales se necesita conservar alguna memoria.
4) Agregue los atributos necesarios para cumplir los requerimientos.

3.3.1.2.1.2. Asociaciones

Es necesario identificar aquellas asociaciones de conceptos que se necesitan para satisfacer los requerimientos de información de los casos de uso bajo creación, y que ayuden en la comprensión del modelo conceptual.

Una asociación es una relación entre conceptos para indicar alguna conexión significativa e interesante. En UML se describen como relaciones estructurales entre objetos de diferentes tipos.

Considere la inclusión de las siguientes asociaciones en un modelo conceptual:

- Asociaciones para las que el conocimiento de la relación necesita preservarse por algún tiempo (asociaciones de "necesita conocer").
- Asociaciones derivadas de la lista de Asociaciones Comunes.

Empiece la adición de asociaciones usando la siguiente lista:

Categoría	*Ejemplos*
A es una parte física de B	Rueda – Camilla, Electrodo – Equipo de Ultrasonido
A es una parte lógica de B	Estudio – Paciente
A esta físicamente contenida en B	Cama – Habitación, Paciente – Hospital
A esta lógicamente contenida en B	Descripción de artículo – Catálogo, Descripción de estado de salud – Expediente
A es una descripción de B	Descripción de artículo – Artículo
A es un artículo de línea de una transición o reporte B	Conteo de glóbulos – Análisis Clínico
A es conocida/anotada/registrada/reportada/ capturada en B	Peso – Expediente de paciente
A es un miembro de B	Médico – Hospital, Laboratorista – Laboratorio de análisis clínicos
A es una subunidad organizacional de B	Subdirección de Enseñanza – Dirección

	General
A usa o administra B	Radiólogo – Unidad de Rayos X
A se comunica con B	Médico – Paciente, Enfermera – Cirujano
A esta relacionado con una transacción B	Paciente – Admisión
A es una transacción relacionada con otra transacción B	Pago – Admisión, Reservación – Cancelación
A esta próximo a B	Quirófano – Quirófano
A es propiedad de B	Tomógrafo – Hospital

De la lista anterior, las categorías de alta prioridad y que son invariablemente útiles en el modelo conceptual son:

- A es parte física o lógica de B.
- A esta contenida física o lógicamente en B.
- A esta registrada en B.

Encontrar conceptos es más importante que encontrar asociaciones. La mayoría del tiempo en la creación del modelo conceptual debe dedicarse a la identificación de conceptos, no de asociaciones.

Recomendaciones para encontrar asociaciones:

- Enfocarse sobre aquellas asociaciones para las cuales el conocimiento de la relación necesita preservarse por algún tiempo.
- Es más importante identificar conceptos que asociaciones.
- Muchas asociaciones tienden a hacer confuso un modelo conceptual en lugar de aclararlo. Su descubrimiento puede ser consumidor de tiempo, con beneficio marginal.
- Evite mostrar asociaciones redundantes o derivables.

Cada extremo de una asociación se llama un *papel*. Los papeles opcionalmente pueden tener: nombre, multiplicidad y navegabilidad.

3.3.1.2.1.3. Atributos

Es fundamental identificar aquellos atributos de conceptos que son necesarios para satisfacer los requerimientos de información de los casos de uso en desarrollo. Recuerde que el modelo conceptual es una representación de cosas del mundo real, no de componentes de software. Algunas declaraciones respecto a los atributos deberán interpretarse en el contexto de entidades del mundo real.

Un atributo es un valor de dato lógico de un objeto. Se deben incluir los siguientes atributos en un modelo conceptual: aquellos para los que los requerimientos (por ejemplo, casos de uso) sugieren o implican la necesidad de recordar información.

Los atributos en el modelo conceptual de preferencia serán atributos simples o valores de datos puros. Como regla de dedo use la prueba básica de tipos de atributo "simple": considere un atributo si en forma natural se piensa de él como un número, cadena, booleano, fecha (y así sucesivamente), de otra manera represéntelo como un concepto separado.

Los atributos no deben usarse para relacionar conceptos en el modelo conceptual. La violación más común de este principio es agregar un tipo de atributo de llave foránea para asociar dos conceptos. Relacione conceptos con una asociación, no con un atributo.

El tipo de un atributo se puede expresar como un tipo no primitivo en el modelo conceptual. Representa lo que inicialmente sea considerado un tipo de dato primitivo como un tipo no primitivo si:

- Esta compuesto de secciones separadas (teléfono, nombre).
- Hay operaciones asociadas con él, tal como analizar o validar.
- Tiene otros atributos.
- Es una cantidad con una unidad.

3.3.1.2.1.4. Comportamiento del sistema – Diagramas de secuencia

Antes de hacer un diseño lógico de cómo trabajará una aplicación, es necesario investigar y definir su comportamiento como una caja negra. El comportamiento de un sistema es una descripción de que hace un sistema, sin explicar como lo hace. Una parte de esta descripción es un diagrama de secuencia. La creación de diagramas de secuencia se da durante la fase de análisis del ciclo de desarrollo.

Un diagrama de secuencia es un dibujo que muestra, para un escenario particular de un caso de uso, los eventos que generan los actores externos, su orden y eventos internos. El tiempo avanza hacia abajo, y el orden de los eventos debe ser el orden en el caso de uso.

El diagrama de secuencia se debe hacer para el curso típico de eventos del caso de uso, y posiblemente para los cursos alternativos más interesantes.

Un evento del sistema es una entrada externa generada por un actor para un sistema. El evento inicia una operación de respuesta. Una operación del sistema es una operación del sistema que ejecuta en respuesta a un evento del sistema. El nombre de un evento y el de la operación son idénticos, la diferencia esta en que el evento es el estímulo y la operación es la respuesta.

Para hacer diagramas de secuencia:

1. Dibuje una línea representando el sistema como una caja negra.
2. Identifique cada actor que opera directamente sobre el sistema. Dibuje una línea para cada actor.
3. Del texto del curso típico de eventos del caso de uso, identifique los eventos (externos) del sistema que genera cada actor. Ilústrelos en el diagrama.
4. Opcionalmente, incluya el texto del caso de uso a la izquierda del diagrama.

Ejemplos de estos diagramas se encuentran en el capítulo 4, en la figuras 4.3, 4.4 y 4.5.

3.3.1.2.1.5. Comportamiento del sistema – Contratos

Un diagrama de secuencia muestra los eventos del sistema que genera un actor externo, pero no muestra los detalles de la funcionalidad asociada con las operaciones del sistema invocadas. Pierde los detalles necesarios para entender la respuesta del sistema.

Un contrato es un documento que describe lo que una operación se compromete a lograr. Usualmente es declarativo en estilo, enfatizando que pasará, en lugar de cómo se logrará. Es común expresarlos en términos de pre y post-condiciones de cambios de estado.

Secciones de un contrato

El contrato esta formado por varias secciones, aunque no todas son necesarias para describir los cambios en el estado del sistema total cuando se invoca una operación del sistema. Se recomienda que el contrato tenga las secciones de Responsabilidades y Post-condiciones. Todas las secciones que puede tener un contrato se muestran en la tabla 3.1.

Para hacer un contrato para cada caso de uso, haga lo siguiente:

1. Identificar las operaciones del sistema a partir de los diagramas de secuencia del sistema.
2. Para cada operación del sistema, construir un contrato.
3. Empezar describiendo la sección de Responsabilidades, describiendo informalmente el propósito de la operación.
4. Después completar la sección Post–condiciones, declarativamente describiendo los cambios de estado que le ocurren a objetos en el modelo conceptual.
5. Para describir las Post – condiciones, use las siguientes categorías:
   - Creación y borrado de instancias

- Modificación de atributo
- Asociaciones formadas y rotas

Tabla 3.1 Secciones de un contrato

Sección	*Descripción*
Nombre	Nombre de la operación y parámetros
Responsabilidades	Una descripción informal de las responsabilidades que debe satisfacer la operación
Tipo	Nombre del tipo (concepto, clase de software, interfase)
Referencias cruzadas	Número de referencia de la función del sistema, casos de uso, etc.
Notas	Notas de diseño, algoritmos, etc.
Excepciones	Casos excepcionales
Salida	Salidas, tales como mensajes o registros que son enviados fuera del sistema.
Pre – condiciones	Suposiciones acerca del estado del sistema antes de ejecutar la operación.
Post - condiciones	El estado del sistema después de completar la operación.

Después de la sección de responsabilidades, la parte más importante del contrato son las Post – condiciones, las cuales establecen como ha cambiado el sistema como resultado de estas operaciones. Las post – condiciones no son acciones para ser ejecutadas durante la operación, sino que son declaraciones acerca del estado del sistema, las cuales son verdaderas cuando la operación ha terminado. Exprese las post – condiciones en tiempo pasado, para enfatizar que son declaraciones acerca de un cambio pasado.

La fase del análisis enfatiza el entendimiento de los requerimientos, conceptos, y operaciones relacionadas con el sistema. Se caracteriza por enfocarse sobre preguntas de cuáles son los procesos, conceptos, etc.

Hay otros artefactos en UML que se pueden usar para capturar los resultados del análisis, sin embargo, con lo que se ha expuesto anteriormente, se tiene el siguiente conjunto mínimo, que es de gran utilidad:

Artefacto del Análisis	*Pregunta contestada*
Casos de uso	¿Cuáles son los procesos del dominio?
Modelo conceptual	¿Cuáles son los términos, conceptos?
Diagramas de secuencia	¿Cuáles son los eventos y operaciones del sistema?
Contratos	¿Qué hacen las operaciones del sistema?

3.3.1.2.2. Diseño

Durante un ciclo de desarrollo iterativo es posible moverse hacia una fase de diseño, una vez que se ha completado el análisis. Durante esta fase, se desarrolla una solución lógica basada en el paradigma orientado a objetos. El corazón de esta solución es la creación de los diagramas de interacción, los cuales ilustran cómo se comunicaran los objetos para satisfacer los requerimientos.

Después de los diagramas de interacción, se pueden dibujar los diagramas de diseño de clases las cuales resumen la definición de las clases (e interfases) que serán implementadas en software.

La creación de diagramas de interacción requiere la aplicación de principios de asignación de responsabilidades y el uso de patrones de diseño.

3.3.1.2.2.1. Casos de uso reales

La definición de casos de uso reales es la primera actividad dentro de la fase de diseño. Su creación depende de los casos de uso esenciales creados con anterioridad.

Un caso de uso real describe el diseño real o efectivo del caso de uso en términos de tecnología de entrada y salida concretos y su implementación total.

3.3.1.2.2.2. Diagramas de interacción

UML incluye los diagramas de interacción para ilustrar cómo interactuan los objetos vía mensajes para cumplir tareas. Su creación es dependiente de la creación anterior de los siguientes artefactos:

- Modelo conceptual – de éste, el diseñador puede elegir definir las clases de software correspondiente a conceptos. Los objetos de estas clases participan en interacciones ilustradas en los diagramas de interacción.
- Contratos de operación – de éstos, el diseñador identifica las responsabilidades y post – condiciones que los diagramas de interacción deben cumplir.
- Casos de uso reales (o esenciales) – de éstos, el diseñador puede tomar información acerca de que tareas cumplen los diagramas de interacción, adicionalmente a las que están en los contratos.

Un diagrama de interacción ilustra las interacciones del mensaje entre instancias (y clases) en el modelo de clase. UML define dos tipos de diagramas de interacción:

- Diagramas de colaboración. Ilustran interacciones de objeto en un formato de gráfica o red.
- Diagramas de secuencia. Ilustran interacciones en un tipo de formato de cerca.

Se deben aplicar los siguientes lineamientos para crear diagramas de colaboración:

1. Crear un diagrama separado para cada operación del sistema en el ciclo de desarrollo actual.
2. Si el diagrama obtenido es complejo (por ejemplo, no cabe en una hoja carta), dividirlo en diagramas más pequeños.
3. Usando las responsabilidades del contrato de operación y las post – condiciones, así como la descripción de casos de uso como punto inicial, diseñe un sistema de objetos interactuando para satisfacer las tareas. Aplique el GRASP y otros patrones para desarrollar un buen diseño.

3.3.1.2.2.3. GRASP: Patrones para asignación de responsabilidades

Debido a que hay una gran variabilidad en la calidad del diseño de interacción de objetos y asignación de responsabilidades, se deben aplicar algunos principios para un buen diseño orientado a objetos. Algunos de estos principios están codificados en los patrones GRASP (General Responsibility Assignment Software Patterns).

Booch y Rumbaugh definen responsabilidad como un contrato u obligación de un tipo o clase. Básicamente estas responsabilidades son de dos tipos:

1. Entendida. Entender acerca de datos privados encapsulados, entender acerca de objetos relacionados, entender acerca de cosas que puede derivar o calcular.

2. Hecha. Hacer algo por si mismo, iniciar una acción en otros objetos, controlar y coordinar actividades en otros objetos.

Una responsabilidad no es lo mismo que un método, sin embargo, los métodos se implementan para satisfacer completamente las responsabilidades. Las responsabilidades se implementan usando métodos, los cuales actúan solos o colaboran con otros métodos y objetos.

Patrones

Los desarrolladores expertos en orientado a objetos han construido un repertorio de soluciones idiomáticas y principios generales que los guían en la creación del software. Estos principios e idiomas, codificados en un formato estructurado que

describe el problema y la solución y a los cuales se les ha dado un nombre, es lo que se puede llamar un patrón. Es decir, un patrón es un par problema-solución con un nombre, el cual se puede aplicar en contextos nuevos, con consejos de cómo aplicarlo en situaciones nuevas.

Los patrones intentan codificar conocimiento, idiomas y principios existentes, y en particular, los patrones GRASP describen principios fundamentales de asignación de responsabilidades a objetos. Es importante entender y ser capaz de aplicar estos principios durante la creación de diagramas de interacción porque un desarrollador de software nuevo en la tecnología de objetos necesita dominar estos principios básicos tan pronto como sea posible; ya que son el fundamento de como se diseñará un sistema. GRASP es un acrónimo para General Responsibility Assignment Software Patterns.

[Larman98] describe cinco patrones que pueden ayudar en esta asignación de responsabilidades: Experto, Creador, Alta Cohesión, Bajo Acoplamiento y Controlador. Aunque existen otros patrones, es conveniente comprender estos primeros cinco patrones porque guían hacia cuestiones comunes, muy básicas y fundamentales en problemas de diseño.

Experto

Solución. Asignar una responsabilidad para el experto en información, la clase que tiene la información necesaria para cumplir la responsabilidad.

Problema. ¿Cuál es el principio más básico por medio del cual se asignan responsabilidades en el diseño orientado a objetos?

Experto se usa más que cualquier otro patrón en la asignación de responsabilidades. Experto no significa una idea fantasiosa u obscura, expresa la intuición común de que los objetos hacen cosas relacionadas con la información que tienen. El cumplimiento de una responsabilidad con frecuencia requiere información que está diseminada en diferentes clases de objetos. Esto implica que hay muchos expertos parciales que colaboraran en la tarea. Experto conduce a diseños donde un objeto de software hace aquellas operaciones las cuales son hechas normalmente por la cosa del mundo real que representa. Este patrón, como muchas cosas en la tecnología de objetos, tiene una analogía con el mundo real. Generalmente damos responsabilidades a individuos que tienen la información necesaria para cumplir la tarea.

Beneficios

- Se mantiene el encapsulamiento, ya que los objetos usan su propia información para cumplir sus tareas. Esto soporta bajo acoplamiento, el cual lleva a sistemas más robustos y mantenibles.

- El comportamiento se distribuye a través de clases que tienen la información requerida, alentando así definiciones de clase ligeras más cohesivas que son más fáciles de entender y mantener. Soporta alta cohesión.

Creador

Solución. Asignar a la clase B la responsabilidad de crear una instancia de clase A si algo de lo siguiente es verdadero:

- B agrega objetos A
- B contiene objetos A
- B registra instancias de objetos A
- B usa estrechamente objetos A
- B tiene los datos inicializados que serán pasados a A cuando se crea (así B es un experto con respecto a la creación de A)

B es un creador de objetos A. Si aplica más de una opción, preferir una clase B la cual agregue o contenga la clase A.

Problema. ¿Quién será responsable de crear una instancia nueva de alguna clase?

Creador conduce a la asignación de responsabilidades relacionadas con la creación de objetos, una tarea muy común en sistemas orientados a objetos. El intento básico de Creador es encontrar un creador que necesita ser conectado con el objeto creado en cualquier evento. Creador sugiere que la clase contenedor o registrador es buena candidata para la responsabilidad de crear la cosa contenida o registrada. El concepto de agregación se usa en el patrón Creador. La agregación involucra cosas que están en una relación fuerte de todo-parte o montaje-parte, tal como Cuerpo agrega Pierna, o Párrafo agrega Oración.

Beneficios

Soporta Bajo Acoplamiento, el cual implica dependencias de mantenimiento más bajas y oportunidades más altas de reuso. El acoplamiento probablemente no se incrementa debido a que la clase creada tal vez ya es visible para la clase creador, debido a las asociaciones existentes que motivaron su elección como creador.

Bajo Acoplamiento

Solución. Asignar una responsabilidad tal que el acoplamiento permanece bajo.

Problema. ¿Cómo soportar dependencia baja e incremento de reuso?

El acoplamiento es una medida de qué tan fuertemente una clase está conectada a, tiene conocimiento de, o depende de otras clases. Una clase con bajo (o débil) acoplamiento no es dependiente de muchas otras clases. Por el contrario, una clase con alto (o fuerte) acoplamiento depende de muchas otras clases y presenta los siguientes problemas:

- Los cambios en las clases con las que se relaciona forza a realizar cambios locales.
- Son difíciles de entender en forma aislada.
- Son difíciles de reusar porque su uso requiere la presencia adicional de las clases de que dependen.

En lenguajes orientados a objetos las formas comunes de acoplamiento del *TipoX* al *TipoY* incluyen:

- El *TipoX* tiene un atributo (dato miembro o variable instancia) que refiere a una instancia *TipoY*, o al mismo *TipoY*.
- El *TipoX* tiene un método el cual referencia una instancia de *TipoY*, o al mismo *TipoY*, por algún medio. Esto típicamente incluye un parámetro o variable local de tipo *TipoY*, o el objeto regresado por un mensaje es una instancia de *TipoY*.
- El *TipoX* es una subclase directa o indirecta de *TipoY*.
- El *TipoY* es una interfase, y el *TipoX* implementa esa interfase.

El Acoplamiento Bajo soporta el diseño de clases que son más independientes, las cuales reducen el impacto de cambios y son más reusables. No se puede considerar en forma aislada de otros patrones, tales como el Experto y el Alta Cohesión, sino que necesita incluirse como uno de varios principios de diseño que influencia una elección en la asignación de una responsabilidad.

El acoplamiento puede no ser importante si el reuso no es una meta. Se deben considerar las metas de reuso en el contexto completo antes de hacer un esfuerzo por minimizar el acoplamiento.

Una subclase está fuertemente acoplada a su superclase. La decisión para derivar desde una superclase necesita ser considerada cuidadosamente ya que es una forma fuerte de acoplamiento.

El caso extremo de Bajo Acoplamiento es cuando hay muy poco o ningún acoplamiento entre clases. Esto no es deseable porque una metáfora central de la tecnología de objetos es un sistema de objetos conectados que se comunican por medio de mensajes. Si se toma con exceso el Bajo Acoplamiento, produce un diseño pobre porque conduce a unos cuantos objetos incohesivos, saturados y complejos que hacen todo el trabajo, con muchos objetos sin acoplamiento muy pasivos que actúan como simple depósito de datos.

Beneficios

- No afectado por cambios en otros componentes.
- Simple de entender en forma aislada.
- Conveniente para reusar.

Alta cohesión

Solución. Asignar una responsabilidad tal que la cohesión permanezca alta.

Problema. ¿Cómo mantener manejable la complejidad?

En términos de diseño orientado a objetos, la cohesión (específicamente, cohesión funcional) es una medida de qué tan fuertemente están relacionadas y enfocadas las responsabilidades de una clase. Una clase con responsabilidades altamente relacionadas, y que no hace una tremenda cantidad de trabajo, tiene alta cohesión. Una clase con baja cohesión hace muchas cosas no relacionadas o hace muchísimo trabajo, tal clase es indeseable, y tiene los siguientes problemas:

- Difícil de comprender
- Difícil de reusar
- Difícil de mantener
- Delicada, afectada constantemente por los cambios

Como el Bajo Acoplamiento, Alta Cohesión es un principio para tener en mente durante todas las decisiones de diseño; es una meta fundamental a considerar continuamente. Como una regla de dedo, una clase con alta cohesión tiene un número relativamente pequeño de métodos, con funcionalidad altamente relacionada, y no hace mucho trabajo. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande.

El patrón de Alta Cohesión, como muchas cosas en la tecnología de objetos, tiene una analogía con el mundo real. Si una persona toma muchísimas responsabilidades no relacionadas, especialmente aquellas que pudieran ser delegadas a otras, entonces esta persona no es efectiva. Esto se observa en algunos administradores que no han aprendido cómo delegar.

Beneficios

- Se incrementa la claridad y facilidad de comprensión del diseño.
- Mantenimiento y mejoras se simplifican.
- Con frecuencia se soporta el bajo acoplamiento.

- El grano fino de funcionalidad altamente relacionada soporta el incremento potencial del reuso porque una clase altamente cohesiva se puede usar para un propósito muy específico.

Controlador

Solución. Asignar la responsabilidad para manejar un mensaje de evento del sistema para una clase representando una de las siguientes elecciones:

- Representa el sistema total (controlador fachada).
- Representa el negocio u organización completa (controlador fachada).
- Representa alguna cosa en el mundo real que es activa (por ejemplo, el papel de una persona) que puede estar involucrada en la tarea (papel controlador).
- Representa un manejador artificial de todos los eventos del sistema de un caso de uso, usualmente llamado "Manejador NombreCasoDeUso" (controlador caso de uso).

Problema. ¿Quién puede ser responsable para manejar un evento del sistema?

Un evento del sistema es un evento de alto nivel generado por un actor externo; es un evento de entrada externo. Están asociados con operaciones del sistema, que son las respuestas a los eventos del sistema. Un Controlador es un objeto de interfase, no de usuario, responsable de manejar un evento del sistema. Un Controlador define el método para la operación del sistema.

La primera categoría de controlador es un controlador fachada representando el sistema total. Pudiera ser algo físico, una clase representando el sistema de software completo, o cualquier otro concepto que el diseñador elija para representar al sistema. Este tipo de controlador es apropiado cuando existen pocos eventos en el sistema, o no es posible redirigir los mensajes de evento del sistema a otros controladores.

Si se aplica la cuarta categoría de controlador, un manejador de caso de uso artificial, entonces hay un controlador diferente para cada caso de uso. Nótese que esto no es un objeto de dominio; es un constructor artificial para soportar al sistema. Es una alternativa a considerar cuando la asignación de responsabilidades, en cualquiera de las otras elecciones, conduce a diseños con baja cohesión o alto acoplamiento. Un controlador caso de uso es buena elección cuando hay muchos eventos a través de diferentes procesos; factoriza su manejo en clases separadas administrables, y también provee una base para razonar acerca del estado del proceso actual.

Beneficios

Potencial incrementado para componentes reusables. Asegura que el negocio o proceso del dominio se maneja por la capa de objetos del dominio en lugar de la capa de interfase. Un diseño de interfase como controlador reduce la oportunidad de reusar la lógica de proceso de dominio en aplicaciones futuras, ya que está limitada a una interfase particular que raramente se usa en otras aplicaciones. En contraste, delegando una responsabilidad de operación del sistema a un controlador entre las clases del dominio soporta el reuso de la lógica para manejar los procesos de negocio relacionados en aplicaciones futuras.

Razón acerca del estado del caso de uso. Algunas veces es necesario asegurar que las operaciones del sistema ocurran en una secuencia legal, o ser capaz para razonar acerca del estado actual de la actividad y operaciones dentro del caso de uso que se recorre.

Patrón Comando

Muchas aplicaciones no tienen una interfase de usuario, sino que reciben mensajes desde algún otro sistema externo; es decir, son sistemas de manejo de mensajes. En una aplicación con una interfase de usuario (por ejemplo, una ventana), la ventana puede elegir quién será el objeto controlador. Diferentes ventanas pueden colaborar con diferentes controladores, especialmente si se usa un controlador de caso de uso.

Para manejar mensajes de evento del sistema en un sistema de manejo de mensajes se debe:

1. Definir un solo controlador para todos los mensajes de evento del sistema; puede ser un controlador de fachada, o un solo controlador del tipo caso de uso llamándole tal vez ManejadorDeMensaje.

2. Usar el patrón Comando para manejar la petición.

El patrón Comando especifica la definición de una clase para cada mensaje o comando, cada uno con un método ejecuta. El controlador creará una instancia Comando correspondiente al mensaje de evento del sistema, y enviará un mensaje ejecuta. Cada clase Comando tiene un método ejecuta único el cual especifica las acciones para el comando.

Patrones Relacionados

- Comando. En un sistema de manejo de mensaje, cada mensaje se puede representar y manejar por un objeto Comando separado.

- Fachada. Eligiendo un objeto que represente a la organización o sistema completo para ser un controlador es un tipo de fachada.

- Retransmisor (Forwarder) - Receptor. Es un patrón útil para sistemas de manejo de mensajes.

- Capas. Colocar la lógica de dominio en la capa de dominio en lugar de en la capa de presentación Capas.

- Fabricación Pura. Es una clase artificial, no un concepto de dominio. Un controlador caso de uso es una clase de Fabricación Pura.

La asignación de responsabilidades y el desarrollo de diagramas de interacción (colaboración y secuencia) es el paso más significativo durante la fase de diseño.

Los pasos en la creación de diagramas de interacción incluyen:
1. Crear diagramas separados para cada operación del sistema. Para cada evento del sistema, hacer un diagrama con él como mensaje inicial.
2. Usando las responsabilidades y post-condiciones del contrato, y la descripción de los casos de uso como punto inicial, diseñar un sistema de objetos interactuantes para satisfacer las tareas. Aplicar GRASP y otros patrones para desarrollar un buen diseño.

La relación entre los artefactos en términos de eventos del sistema son:
- Los casos de uso sugieren los eventos del sistema los cuales se muestran explícitamente en diagramas de secuencia.
- La mejor conjetura inicial en el efecto de los eventos del sistema se describe en los contratos de operación del sistema.
- Los eventos del sistema representan mensajes que inician diagramas de interacción, los cuales ilustran cómo interactuan los objetos para satisfacer las tareas requeridas.
- Los diagramas de interacción involucran interacción de mensajes entre objetos definidos en el modelo conceptual, más otras clases de objetos.

3.3.1.2.2.4. Diseño de diagramas de clase

El diseño de diagramas de clase ocurre dentro de la fase de diseño de un ciclo de desarrollo. Su creación depende de la creación anterior de:

- Diagramas de interacción. De éstas, el diseñador identifica las clases de software que participan en la solución, más los métodos de las clases.
- Modelo conceptual. De éste, el diseñador agrega detalle a las definiciones de clase

Un diseño de diagramas de clase ilustra las especificaciones para clases de software e interfases en una aplicación. La información típica que se incluye es la siguiente:

- Clases, asociaciones y atributos
- Interfases con sus operaciones y constantes
- Métodos
- Información del tipo de atributo
- Navegabilidad
- Dependencias

La estrategia a aplicar para crear un diagrama de clases es la siguiente:

1. Identificar todas las clases que participan en la solución de software. Hacer esto analizando los diagramas de interacción.
2. Dibujarlas en un diagrama de clases.
3. Duplicar los atributos de los conceptos asociados en el modelo conceptual.
4. Agregar nombres de métodos analizando los diagramas de interacción.
5. Agregar información de tipo para los atributos y métodos.
6. Agregar las asociaciones necesarias para soportar la visibilidad de atributo requerida.
7. Agregar flechas de navegabilidad para las asociaciones para indicar la dirección del atributo de visibilidad.
8. Agregar líneas de relaciones de dependencia para indicar visibilidad que no es de atributo.

Los métodos de cada clase se pueden identificar analizando los diagramas de colaboración. En general, el conjunto de todos los mensajes enviados a una clase X a través de todos los diagramas de colaboración indica la mayoría de métodos que la clase X debe definir.

Cada extremo de una asociación se llama un papel, y en un diagrama de clase puede decorarse con una flecha de navegabilidad. La navegabilidad es una propiedad del papel que indica que es posible navegar unidireccionalmente a través de la asociación de los objetos de la clase fuente a la destino. La navegabilidad implica visibilidad, usualmente de atributo.

UML incluye una relación de dependencia general la cual indica que un elemento tiene conocimiento de otro elemento. Se ilustra con una línea con flecha punteada. En un diagrama de clase la relación de dependencia es útil para dibujar visibilidad que no es de atributos entre clases; en otras palabras, visibilidad declarada como

parámetro, global o localmente. En contraste, la visibilidad de atributo plana se muestra con una línea de asociación regular y una flecha de navegabilidad.

3.3.1.2.3. Construcción

Al terminar el diseño del diagrama de clases en el ciclo de desarrollo actual, hay suficiente detalle para generar código para la capa de dominio de los objetos, que corresponde a la del dominio del problema y los objetos representan los conceptos del dominio, tal como un mensaje DICOM.

Los artefactos de UML, los diagramas de colaboración y de clase, creados durante la fase de diseño se usaran como entradas al proceso de generación de código. El orden y ocurrencia de las actividades de la fase de construcción pueden variar mucho, sin embargo, un ejemplo de actividades posible es el siguiente:

- Definiciones de Interfase e Implementación de Clases
- Implementación de Métodos
- Implementación de Ventanas
- Implementación de Reportes
- Implementación del esquema de la Base de Datos
- Escribir Código de Prueba

Para reducir el riesgo y aumentar la probabilidad de desarrollar una aplicación apropiada, el desarrollo deberá estar basado en una cantidad significativa de análisis y diseño antes de empezar la codificación. Esto no sugiere que no hay espacio para prototipos o diseños mientras se programa. Sin embargo, el núcleo de la aplicación, tal como el modelo conceptual, las capas arquitecturales, asignación de responsabilidades, e interacción de objetos se determinan mejor en un proceso de investigación y diseño formal en lugar de apresurarse a codificar. La precipitación a codificar crea sistemas difíciles de entender, extender y mantener, y no soporta un proceso repetible exitoso.

Algunas veces es mejor interrumpir la fase de diseño para hacer alguna programación de exploración para descubrir un diseño factible, y después regresar a la fase formal de diseño.

La creación de código en un lenguaje de programación orientado a objetos no es parte del análisis y diseño orientado a objetos; es una meta final. Los artefactos creados en la fase de diseño proporcionan un grado significativo de la información necesaria para generar el código, lo que es un proceso de traducción relativamente directo.

Una fortaleza del análisis, diseño y programación orientada a objetos, cuando se usa con un proceso de desarrollo, es que proporciona un mapa completo de extremo a extremo desde los requerimientos hasta el código. Esto no sugiere que el camino será fácil, o que puede simplemente seguirse de manera mecánica,

existen muchas variables. Pero tener un mapa proporciona un punto de inicio para la experimentación y la discusión.

En general la fase de programación no es un paso trivial de generación de código, sino totalmente lo opuesto. Realmente, los resultados generados durante el diseño son un primer paso incompleto; durante la programación y prueba una cantidad muy grande de cambios se hará y los problemas detallados serán descubiertos y resueltos.

Bien hechos, los artefactos de diseño proporcionaran un núcleo resistente que se adapta con elegancia y robustez para satisfacer los nuevos problemas encontrados durante la programación. Consecuentemente, espere y planee cambios y desviaciones del diseño durante la fase de construcción y prueba.

La implementación en un lenguaje orientado a objetos requiere escribir código fuente para: las definiciones de clase y las definiciones de método. Como mínimo, el diseño de diagramas de clase describe el nombre de la clase, superclases, signado de métodos, y atributos simples de una clase. Esto es suficiente para crear una definición de clase básica en un lenguaje de programación orientado a objetos.

Un atributo de referencia es un atributo que se refiere a otro objeto complejo, no a un tipo primitivo como un número o cadena. Estos atributos de referencia de una clase son sugeridos por las asociaciones y navegabilidad en un diagrama de clases. Los atributos de referencia de una clase con frecuencia son implicados, no explícitos, en el diseño del diagrama de clases. Esto es, no se declara explícitamente como un atributo en la sección de atributos de la clase. Existe una visibilidad de atributo sugerida, indicada por la asociación y navegabilidad, lo cual se define explícitamente como un atributo durante la fase de generación de código.

Cada extremo de una asociación se llama un rol, un nombre de rol identifica el rol y proporciona algún contexto semántico como la naturaleza del rol. Si el nombre del rol esta presente en el diagrama de clases, úselo como el nombre del atributo de referencia durante la generación de código.

Un diagrama de colaboración muestra los mensajes que se envían en respuesta a una invocación de método. La secuencia de estos métodos se traduce a una serie de declaraciones en la definición del método. Cada mensaje secuencial dentro de un método, como se muestra en el diagrama de colaboración, se mapea a una declaración.

Con frecuencia es necesario para un objeto mantener la visibilidad a un grupo de otros objetos; la necesidad para ésto usualmente es evidente del valor de multiplicidad en un diagrama de clases. En lenguajes de programación orientados a objetos estas relaciones se implementan frecuentemente con la introducción de un contenedor o colección intermedia. La clase del lado de multiplicidad uno define

un atributo de referencia apuntando a una instancia contenedor/colección, la cual contiene instancias de la clase del lado de multiplicidad de muchos.

Las clases necesitan ser implementadas de la menos acoplada a la más acoplada, por ejemplo aquellas clases que sólo dependen de implementaciones anteriores, y que a su vez de ellas no depende ninguna otra clase.

3.3.1.2.4. Prueba

Aunque la fase de prueba se muestra como un paso final dentro de un ciclo de desarrollo, se recomienda como una actividad en curso durante la fase de Construcción. No todas las pruebas sugeridas en la figura 3.11 son apropiadas durante cada ciclo de desarrollo. Por ejemplo, la prueba de integración del sistema total puede ser menos frecuente en cada ciclo de desarrollo.

3.3.1.3. Entrega (Deploy)

La fase de entrega involucra todo aquello que debe hacerse para poner el sistema en producción como se muestra en la figura 3.12. Para el software comercial esto significa venderlo y que lo usen los clientes; para el software interno esto significa ponerlo en operación. El orden y ocurrencia de las actividades en esta fase varían mucho, para los sistemas de software comercial grandes puede incluir tareas extraordinarias como pruebas de aceptación por miles de usuarios y el establecimiento de grandes centros de soporte. [Larman98] menciona que la discusión de esta fase queda fuera de su alcance.

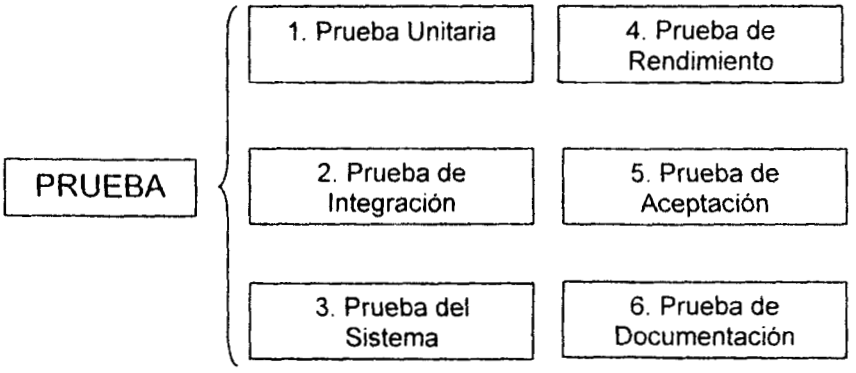

Figura 3.11 Actividades en la fase de Prueba

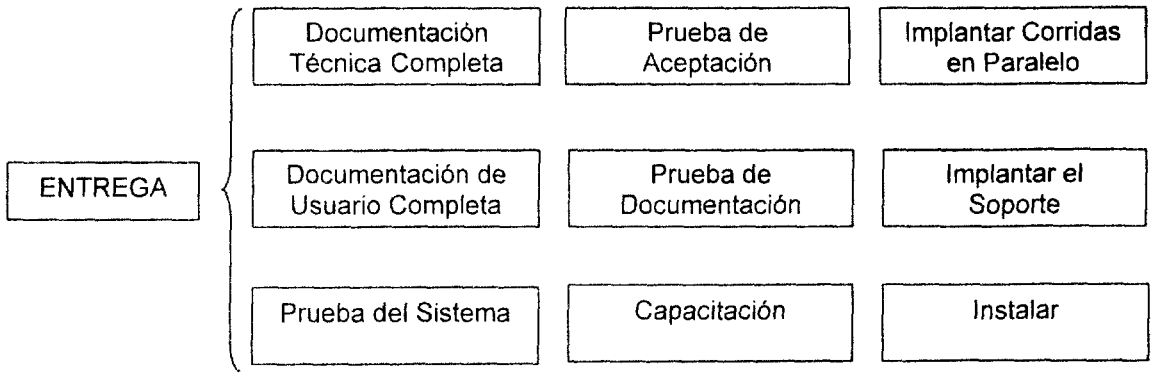

Figura 3.12 Actividades en la fase de Entrega del Sistema

4. Propuesta de solución

Tratando de seguir el proceso de desarrollo descrito en el capítulo anterior, a continuación presento algunos resultados obtenidos al aplicar dicho proceso agrupados según los pasos principales del mismo.

4.1. Planear y Elaborar

4.1.1. Requerimientos

a).- Declaración descriptiva.
- El propósito de este proyecto es crear el sistema del Elemento de Servicio de Mensaje DICOM (DIMSE).

b).- Clientes.
- El presente proyecto no surge de una necesidad en alguna empresa o negocio, sino como proyecto de tesis en un postgrado.

c).- Metas.
- Proveer los servicios que se requieren para que un par de entidades de aplicación puedan intercambiar mensajes entre ellas.

- Construir el siguiente nivel al Protocolo de Capa Superior, el cual es la parte con que ha iniciado un proyecto mayor que es el construir un PACS en el área de Procesamiento Digital de Señales e Imágenes Biomédicas (PDSIB) de la Universidad Autónoma Metropolitana.

d).- Funciones del sistema. El sistema debe:

- Soportar la comunicación entre pares de usuarios de servicio DIMSE.
- Recibir primitivas de petición del invocador.
- Emitir primitivas de indicación al ejecutor.
- Recibir primitivas de respuesta del ejecutor.
- Emitir primitivas de confirmación al invocador.
- Proveer dos tipos de servicios de transferencia de información: notificación y operación.
- Proporcionar los servicios para almacenar, comparar, recuperar y mover información de Instancias SOP Compuestas, así como el servicio de verificación de comunicación entre usuarios de DIMSE.
- Proporcionar los servicios para recuperar y modificar información, así como ejecutar una acción, crear y borrar una instancia de clase, y reportar eventos.
- Intercambiar solicitudes y respuestas de comando entre pares de usuarios de servicio DIMSE.
- Construir mensajes.

- Aceptar mensajes y pasarlos al usuario de servicio DIMSE por medio de primitivas de servicio de indicación y confirmación.
- Enviar mensajes.

e).- Atributos del sistema.
- El sistema se desarrollará para plataforma UNIX, específicamente en IRIX de Silicon Graphics.
- Para las pruebas del sistema, su interfase será en modo texto.

4.1.2. Casos de uso

La frontera del sistema de la aplicación del Elemento de Servicio de Mensaje DICOM (DIMSE), será el hardware y software de la misma, cuyos actores (agentes externos) son: por parte de la Entidad de Aplicación, un Usuario de Clase de Servicio (SCU) o invocador y un Proveedor de Clase de Servicio (SCP) o ejecutor, y por parte del Servicio de capa Superior OSI, al protocolo de Capa Superior DICOM (DULP).

La siguiente tabla muestra a los actores y los procesos que ellos inician:

Actor	Proceso
SCU	Solicitar y Cancelar servicios DIMSE
SCP	Responder solicitudes de servicio
DULP	Transferir mensaje

Considerando que los casos de uso son secuencias de eventos de un actor al usar un sistema para completar un proceso, es decir, son historias o casos de cómo usar un sistema, entonces la solicitud y cancelación de cada uno de los servicios que debe proporcionar DIMSE, es un caso de cómo se debe usar el sistema, otro caso de cómo usar el sistema es emitir primitivas de respuesta con estado igual a pendiente, indicando que se ha iniciado un proceso de comparación de atributos. Existe además otro caso de uso adicional en el sistema, y es para enviar mensajes. Los casos de uso del sistema son los siguientes:

- Proveer Servicios DIMSE
- Respuesta Pendiente
- Cancelar Respuesta Pendiente
- Intercambiar Mensaje

La identificación de los casos de uso antes mencionados es el resultado de realizar varias iteraciones en el proceso de desarrollo basado en un crecimiento y refinamiento sucesivo a través de varios ciclos.

Casos de Uso en Formato de Alto Nivel

La descripción de los casos de uso en un formato de alto nivel, tiene como finalidad el poder expresar, de una manera rápida, lo que se ha comprendido de la complejidad y funcionalidad del sistema en estudio. A continuación se explica en forma breve cada uno de ellos.

Caso de uso:	Proveer Servicios DIMSE
Actores:	SCU (invocador), SCP (ejecutor), DULP
Tipo:	Primario
Descripción:	Un usuario de servicio DIMSE, llamado invocador, solicita a DIMSE un servicio para intercambiar un mensaje con otro usuario de servicio DIMSE, llamado ejecutor. Cuando este último recibe el mensaje, realiza alguna operación de acuerdo al mensaje recibido y envía una respuesta con estado de éxito o error al invocador, terminando así el intercambio del mensaje.

Caso de uso:	Respuesta Pendiente
Actores:	SCP (invocador), SCU (ejecutor), DULP
Tipo:	Primario
Descripción:	Un usuario de servicio DIMSE, llamado invocador, solicita a DIMSE un servicio para intercambiar un mensaje con otro usuario de servicio DIMSE, llamado ejecutor. Cuando éste último recibe el mensaje, realiza alguna operación de acuerdo al mensaje recibido y notifica al invocador que la operación está en proceso por medio de una respuesta con estado igual a pendiente. El intercambio del mensaje se termina cuando el ejecutor envía una respuesta con estado de Éxito, Fallo o Rechazo cuando no puede realizar la operación o bien no la soporta.

Caso de uso:	Cancelar Respuesta Pendiente
Actores:	SCP (invocador), SCU (ejecutor), DULP
Tipo:	Primario
Descripción:	Un usuario de servicio DIMSE, llamado invocador, solicita a DIMSE un servicio para cancelar el intercambio de un mensaje cuando otro usuario de servicio DIMSE, llamado ejecutor, esta emitiendo repuestas con estado igual a pendiente. Cuando este último recibe la petición de cancelación, notifica al invocador que la operación se ha cancelado por medio de una respuesta con estado igual a cancelado, terminando así el intercambio del mensaje.

Caso de Uso:	Intercambiar Mensaje
Actores:	DIMSE (iniciador), DULP
Tipo:	Primario
Descripción:	El sistema DIMSE utiliza los servicios de DULP para intercambiar un mensaje entre un invocador y un ejecutor de servicios DIMSE.

Diagrama de casos de uso

El diagrama de casos de uso ilustra el conjunto de casos de uso para el sistema en desarrollo, los actores, y la relación entre actores y casos de uso. Este diagrama presenta un tipo de diagrama de contexto en el cual se muestran los actores externos del sistema, y la manera en que estos actores usan los casos de uso. Por ejemplo, el caso más general (Proveer Servicios DIMSE) lo inicia el actor SCU como invocador de un servicio, sin embargo, este mismo caso de uso también se relaciona con el SCP como ejecutor del servicio, ya que tiene que dar una respuesta sobre el servicio solicitado. Por otro lado, este mismo caso de uso tiene una relación de "uso" con el caso de uso Intercambiar mensaje, puesto que Proveer Servicios DIMSE usa a Intercambiar mensaje para realizar el intercambio de mensajes entre los dos actores.

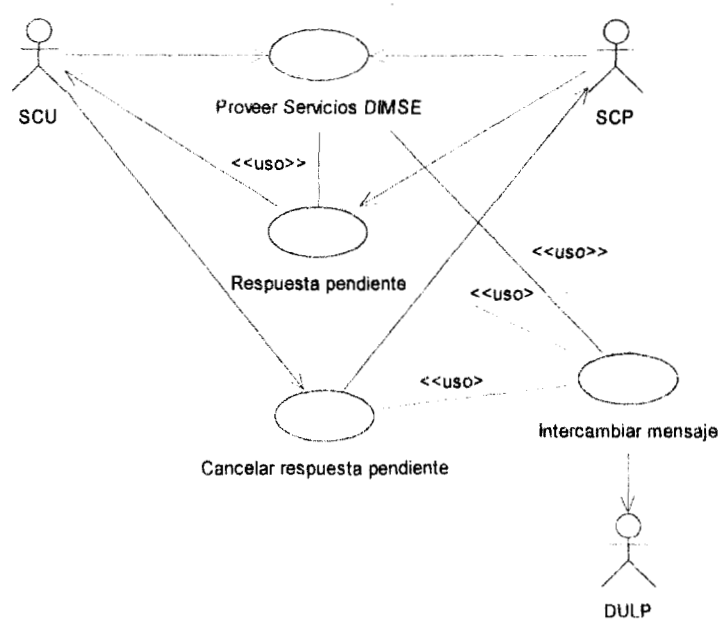

Figura 4.1 Diagrama de Casos de Uso

Casos de uso extendidos

El formato extendido de los casos de uso agrega más detalle a la descripción de los mismos para aclarar y mejorar el entendimiento de ellos. Se incluye una sección de Curso típico de eventos, la cual describe las secuencias de los eventos en un proceso normal, los eventos raros, atípicos o excepcionales se describen en una sección de Cursos alternativos. Se incluye además una clasificación de los casos de uso, considerándolos como primarios si representan procesos comunes mayores, secundarios si representan procesos raros o menores, y opcionales si representan procesos que no necesariamente deben considerarse, además el caso de uso es esencial si esta relativamente libre de detalles de implementación

.y tecnología, describiendo el proceso en términos de sus actividades esenciales y motivación.

Caso de uso:	Proveer Servicios DIMSE
Actores:	SCU (invocador), SCP (ejecutor), DULP
Propósito:	Solicitar un servicio DIMSE y recibir la respuesta correspondiente
Descripción:	Un usuario de servicio DIMSE, llamado invocador, solicita a DIMSE un servicio para intercambiar un mensaje con otro usuario de servicio DIMSE, llamado ejecutor. Cuando este último recibe el mensaje, realiza alguna operación de acuerdo al mensaje recibido y envía una respuesta al invocador, terminando así el intercambio del mensaje.
Tipo:	Primario y esencial.

Curso típico de eventos:

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1.- Este caso de uso empieza cuando un invocador solicita a DIMSE un servicio para pedir a un ejecutor que realice una operación.	
2.- El invocador emite una primitiva de petición de servicio al proveedor de servicio DIMSE, es decir, al sistema en cuestión.	3.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP.
	4.- Al recibir el mensaje de petición, emite una primitiva de indicación de servicio para el ejecutor.
5.- El ejecutor recibe la primitiva de indicación del proveedor de servicio DIMSE.	
6.- El ejecutor reporta el resultado de la operación solicitada emitiendo una primitiva de respuesta de servicio al proveedor de servicio DIMSE.	7.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP.
	8.- Al recibir el mensaje de respuesta, emite una primitiva de confirmación de servicio para el invocador.
9.- El invocador recibe la primitiva de confirmación del proveedor de servicio DIMSE, completándose así la operación y terminando el intercambio del mensaje.	

Cursos alternativos:

Punto 6.- El SCP puede enviar una primitiva de respuesta con estado de Falló o Rechazado antes de recibir en forma completa la indicación del proveedor de Servicio DIMSE.

Caso de uso:	Respuesta Pendiente
Actores:	SCP (invocador), SCU (ejecutor), DULP
Propósito:	Solicitar un servicio DIMSE y recibir más de una respuesta
Descripción:	Un usuario de servicio DIMSE, llamado invocador, solicita a DIMSE un servicio para intercambiar un mensaje con otro usuario de servicio DIMSE, llamado ejecutor. Cuando este último recibe el mensaje, realiza alguna operación de acuerdo al mensaje recibido y notifica al invocador que la operación esta en proceso por medio de una respuesta con estado igual a pendiente. El intercambio del mensaje se termina cuando el ejecutor envía una respuesta con estado de Éxito, Fallo o Rechazado.
Tipo:	Primario y esencial.

Curso típico de eventos:

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1.- Este caso de uso empieza cuando un invocador solicita a DIMSE un servicio para pedir a un ejecutor que realice una operación.	
2.- El invocador emite una primitiva de petición de servicio al proveedor de servicio DIMSE, es decir, al sistema en cuestión.	3.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP.
	4.- Al recibir el mensaje de petición, emite una primitiva de indicación de servicio para el ejecutor.
5.- El ejecutor recibe la primitiva de indicación del proveedor de servicio DIMSE.	
6.- El ejecutor reporta el resultado de la operación solicitada emitiendo una o más primitivas de respuesta de servicio al proveedor de servicio DIMSE, notificando que la operación se encuentra en proceso.	7.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP.
	8.- Al recibir el mensaje de respuesta, emite una primitiva de confirmación de servicio en proceso para el invocador.
9.- El invocador recibe la primitiva de confirmación del proveedor de servicio DIMSE, quedando en espera de más confirmaciones.	
10.- Cuando el ejecutor termina la operación, reporta el resultado de la misma emitiendo una primitiva de respuesta final.	11.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP.
	12.- Al recibir el mensaje de respuesta final, emite una primitiva de confirmación de servicio para el invocador.
13.- El invocador recibe la primitiva de confirmación final del proveedor de servicio DIMSE, completándose así la operación y terminando el intercambio del mensaje.	

Caso de uso:	Cancelar Respuesta Pendiente
Actores:	SCP (invocador), SCU (ejecutor), DULP
Propósito:	Solicitar la cancelación de un servicio DIMSE en proceso
Descripción:	Un usuario de servicio DIMSE, llamado invocador, solicita a DIMSE un servicio para cancelar el intercambio de un mensaje cuando otro usuario de servicio DIMSE, llamado ejecutor, esta emitiendo repuestas con estado igual a pendiente. Cuando este último recibe la petición de cancelación, notifica al invocador que la operación se ha cancelado por medio de una respuesta con estado igual a cancelado, terminando así el intercambio del mensaje.
Tipo:	Primario y esencial.

Curso típico de eventos:

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1.- Este caso de uso empieza cuando un invocador solicita a DIMSE un servicio para pedir a un ejecutor que realice una operación.	
2.- El invocador emite una primitiva de petición de servicio al proveedor de servicio DIMSE, es decir, al sistema en cuestión.	3.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP. 4.- Al recibir el mensaje de petición, emite una primitiva de indicación de servicio para el ejecutor.
5.- El ejecutor recibe la primitiva de indicación del proveedor de servicio DIMSE.	
6.- El ejecutor reporta el resultado de la operación solicitada emitiendo una o más primitivas de respuesta de servicio al proveedor de servicio DIMSE, notificando que la operación se encuentra en proceso.	7.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP. 8.- Al recibir el mensaje de respuesta, emite una primitiva de confirmación de servicio en proceso para el invocador.
9.- El invocador recibe la primitiva de confirmación del proveedor de servicio DIMSE, quedando en espera de más confirmaciones.	
10.- El invocador solicita a DIMSE la cancelación del servicio en ejecución.	11.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP. 12.- Al recibir el mensaje de petición, emite una primitiva de indicación de cancelación de servicio para el ejecutor.
13.- El ejecutor termina la operación, reportando la cancelación de la misma emitiendo una primitiva de respuesta final.	14.- Construye el mensaje y lo envía usando el servicio P-DATA de DULP.

15.- Al recibir el mensaje de respuesta final, emite una primitiva de confirmación de servicio para el invocador.

16.- El invocador recibe la primitiva de confirmación de cancelación del proveedor de servicio DIMSE, terminando el intercambio del mensaje.

Caso de uso:	Intercambiar Mensaje
Actores:	DIMSE (iniciador), DULP
Propósito:	Fragmentar el mensaje y enviarlo sobre una asociación dada.
Descripción:	El sistema DIMSE utiliza los servicios de DULP para intercambiar un mensaje entre un invocador y un ejecutor de servicios DIMSE.
Tipo:	Primario y esencial.

Curso típico de eventos:

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
	1.- Este caso de uso empieza cuando DIMSE encapsula los mensajes DICOM en un grupo de comandos y un grupo de datos.
	2.- DIMSE fragmenta un mensaje DICOM en Fragmentos de Comando y Fragmentos de Datos, colocando cada uno en una unidad PDV.
	3.- DIMSE solicita el servicio P-DATA a DULP para enviar la serie resultante de PDVs.
4.- DULP envía los PDVs sobre una asociación previamente establecida, preservando el orden de los fragmentos.	
	5.- El mensaje termina de enviarse cuando se han enviado todos los fragmentos del mismo.

4.2. Construir

4.2.1. Análisis

Modelo conceptual

La identificación de los casos de uso es la parte final de la fase de Planear y Elaborar y debe empezarse la fase de Construir en la cual se dan los ciclos de desarrollo. Como se mencionó anteriormente, las actividades más importantes del ciclo de desarrollo son el Análisis y el Diseño, y como primera actividad dentro del análisis esta la construcción de un modelo conceptual.

El primer paso para hacer un modelo conceptual es hacer una lista de los conceptos candidatos para el dominio de DIMSE:

- Mensaje Invocador Ejecutor DULP Grupo de Comando
- Petición Indicación Respuesta Confirmación Grupo de Datos
- Servicio Proveedor PDV Asociación Elemento de Comando

Con estos conceptos candidatos, y estableciendo las relaciones entre ellos se establece el siguiente modelo conceptual, cuya finalidad es ayudar a entender como se hace el intercambio de mensajes en DIMSE.

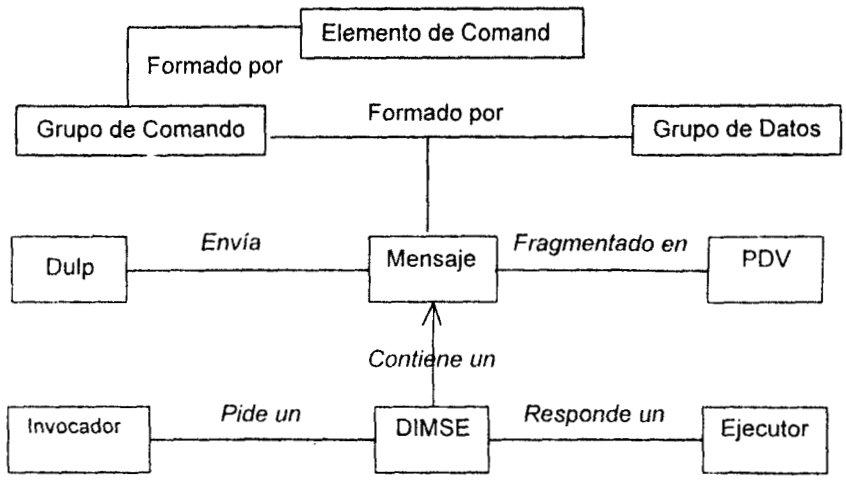

Figura 4.2 Modelo conceptual de DIMSE

Atributos

Los servicios DIMSE-C permiten a una Entidad de Aplicación DICOM explícitamente solicitar una operación a otra entidad de aplicación sobre Instancias SOP Compuestas. Los servicios de este tipo, así como sus atributos se muestran en la siguiente tabla:

DIMSE

Tabla 4.1 Servicios DIMSE-C

	C-STORE		C-FIND			C-GET			C-MOVE			C-ECHO	
Nombre del Parámetro DIMSE-C	Pet	Res	Pet	Res	Can	Pet	Res	Can	Pet	Res	Can	Pet	Res
Mensaje ID	M	-	M	-	-	M	-	-	M	-	-	M	-
Mensaje ID siendo respondido a	-	M	-	M	M	-	M	M		M	M	-	M
Clase SOP Afectada UID	M	U(=)	M	U(=)	-	M	U(=)	-	M	U(=)	-	M	U(=)
Instancia SOP Afectada UID	M	U(=)											
Prioridad	M	-	M	-	-	M	-	-	M	-	-		
Destino de mover									M	-	-		
Identificador			M	C	-	M	U	-	M	U	-		
Estado (status)	-	M	-	M	-	-	M	-	-	M	-	-	M
Título de Entidad de Aplicación del que Origina el Mover	U	-											
Mensaje ID del que Origina el Mover	U	-											
Conjunto de Datos	M	-											
Número de sub-operaciones restantes						-	c	-	-	c	-		
Número de sub-operaciones completadas						-	c	-	-	c	-		
Número de sub-operaciones falladas						-	c	-	-	c	-		
Número de sub-operaciones con precaución						-	c	-	-	c	-		

Pet = Petición
Res = Respuesta
Can = Cancelación
(=) El valor del parámetro es igual al valor del parámetro en la columna a la izquierda
C El parámetro es condicional
M Uso obligatorio
U El uso de este parámetro es una opción del usuario de servicios DIMSE
- No aplicable

Los servicios DIMSE-N soportan operaciones asociadas con Instancias SOP normalizadas y proporcionan un conjunto extendido de operaciones y notificaciones orientadas a objetos. Los servicios de este tipo, así como sus atributos se muestran en la siguiente tabla:

Tabla 4.2 Servicios DIMSE-N

	N-EVENT-REPORT		N-GET		N-SET		N-ACTION		N-CREATE		N-DELETE	
Nombre del Parámetro DIMSE-N	Pet	Res	Pet	Res	Pet	Res	Pet	Res	Pet	Res	Pet	Res
Mensaje ID	M	-	M	-	M	-	M	-	M	-	M	-
Mensaje ID siendo respondido a	-	M	-	M	-	M	-	M	-	M	-	M
UID de Clase SOP Afectada	M	U(=)	-	U	-	U	-	U	M	U(=)	-	U
UID de Instancia SOP Afectada	M	U(=)	-	U	-	U	-	U	U	C	-	U
ID de Tipo de Evento	M	C(=)										
Información de Evento	U	-										
Contestar Evento	-	C										
Estado	-	M	-	M	-	M	-	M	-	M	-	M
UID de Clase SOP Solicitada			M	-	M	-	M	-			M	-
UID de Instancia SOP Solicitada			M	-	M	-	M	-			M	-
Lista de Identificador de Atributo			U	-								
Lista de Atributo			-	C	-	U			M	U		
Lista de Modificación					M	-						
ID de Tipo de Acción							M	C(=)				
Información de Acción							U	-				
Contestar Acción							-	C				

Los parámetros para los servicios de DIMSE que se muestran en las tablas anteriores se presentan solamente con fines informativos, para que se tenga una idea de lo que se debe manejar en la prestación de estos servicios. En esta parte del proceso únicamente se tiene interés en la representación de cosas del mundo real y no en los componentes de software. A nivel de modelo conceptual solo se identifican como atributos el servicio, que debe ser proporcionado por el concepto Dimse, y los parámetros que requiere el servicio. Los parámetros son el atributo del concepto Mensaje.

Diagramas de secuencia

Un diagrama de secuencia ilustra los eventos de los actores hacia el sistema, su creación depende de los casos de uso descubiertos anteriormente, e intenta describir lo que hace el sistema sin explicar como lo hace, es decir, pretende

modelar el comportamiento del sistema. Su creación ocurre durante la fase de análisis de un ciclo de desarrollo.

El diagrama de secuencia de un sistema es un retrato que muestra, para un escenario particular de un caso de uso, los eventos que generan los actores externos, su orden, y eventos internos del sistema. Los sistemas se tratan como cajas negras y el énfasis de los diagramas esta en los eventos que cruzan la frontera del sistema desde los actores hacia los sistemas.

En la figura 4.3 se muestra el curso de eventos particular para el caso de uso Proveer Servicios DIMSE. Como se observa en la figura, el actor externo invocador, solicita un servicio al proveedor Dimse1, este a su vez construye el mensaje de petición correspondiente, y lo envía utilizando el servicio P-DATA del protocolo DULP. Del lado del ejecutor, al recibirse el mensaje se pasa al ejecutor como una indicación de servicio. Una vez realizada la operación asociada con el servicio, el ejecutor utiliza una primitiva de respuesta de servicio para informar al invocador sobre el resultado de la operación solicitada. El proveedor de servicio Dimse2 construye el mensaje de respuesta, y lo envía usando nuevamente al protocolo DULP. Al recibirse el mensaje de respuesta del lado del invocador, el proveedor de servicio Dimse1 le confirma el servicio que solicito, terminando así el proceso de intercambio de mensaje.

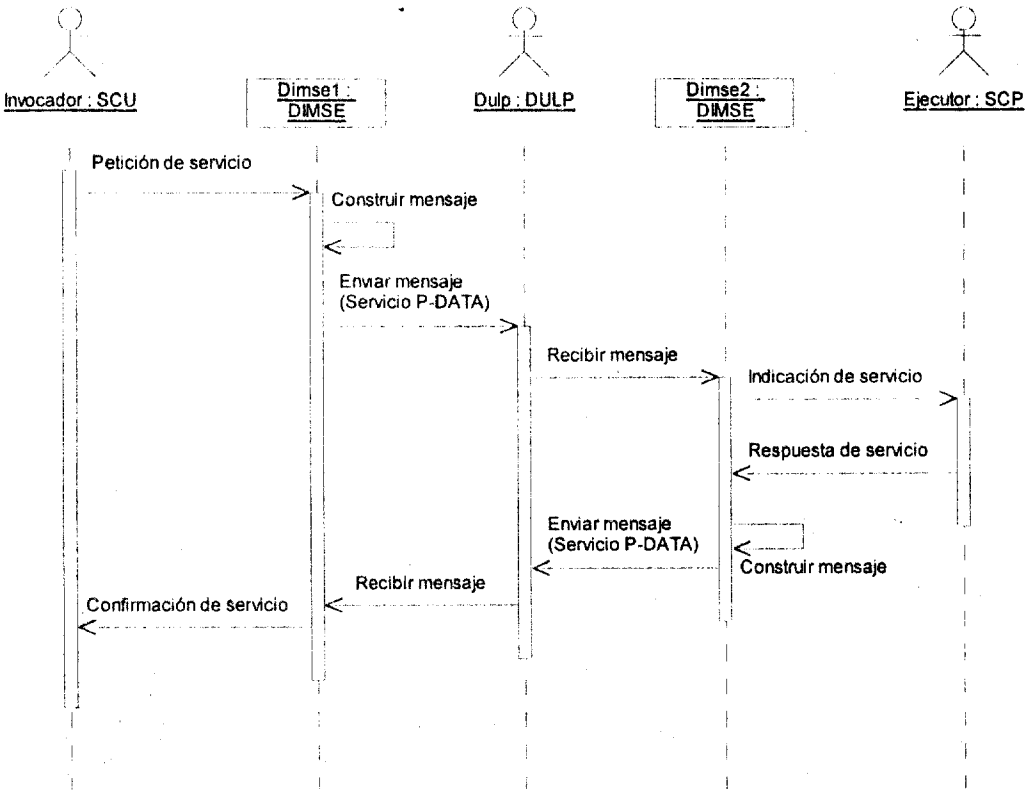

Figura 4.3 Diagrama de secuencias para el caso de uso Proveer Servicios DIMSE

Este diagrama de secuencia también se cumple cuando el ejecutor envía una respuesta temprana, es decir, antes de que los datos de la petición terminen de recibirse por completo, notificando que el servicio se ha rechazado o hubo algún problema por el cual se termina el servicio.

El comportamiento del caso de uso Respuesta Pendiente, mostrado en la figura 4.4, es similar al del caso de uso Proveer Servicios DIMSE, lo que lo distingue de este último es que el ejecutor genera más de una primitiva de respuesta, esto es, emite una o más respuestas informando que el proceso se esta ejecutando y que aun no se termina, incluyendo en estas respuestas un estado de pendiente. Al completar el proceso, envía una respuesta final notificando que el proceso se ha concluido.

Finalmente, la figura 4.5 muestra el comportamiento del caso de uso Cancelar Respuesta Pendiente, a través de su respectivo diagrama de secuencia. Nuevamente se observa una similitud entre este diagrama y el de la figura 4.4, lo que los hace diferentes, es que cuando el ejecutor esta enviando respuestas para indicar que el proceso esta en ejecución, y antes de que se reciba una respuesta final, el invocador puede solicitar la cancelación de la ejecución del proceso mencionado, a lo que el ejecutor responde con una respuesta final, confirmando la cancelación y dando por terminado el intercambio del mensaje.

DIMSE

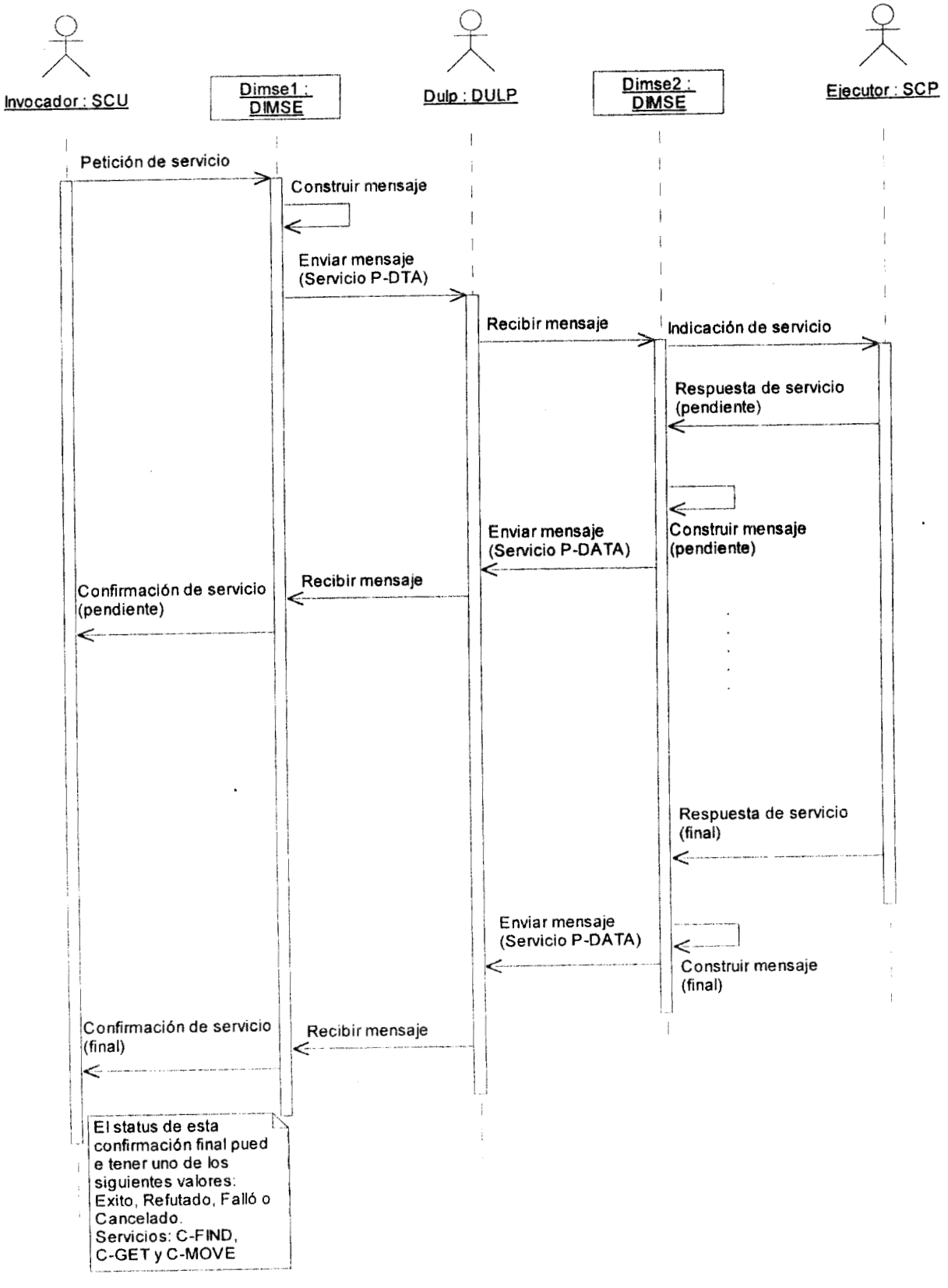

Figura 4.4 Diagrama de secuencia para el caso de uso Respuesta Pendiente

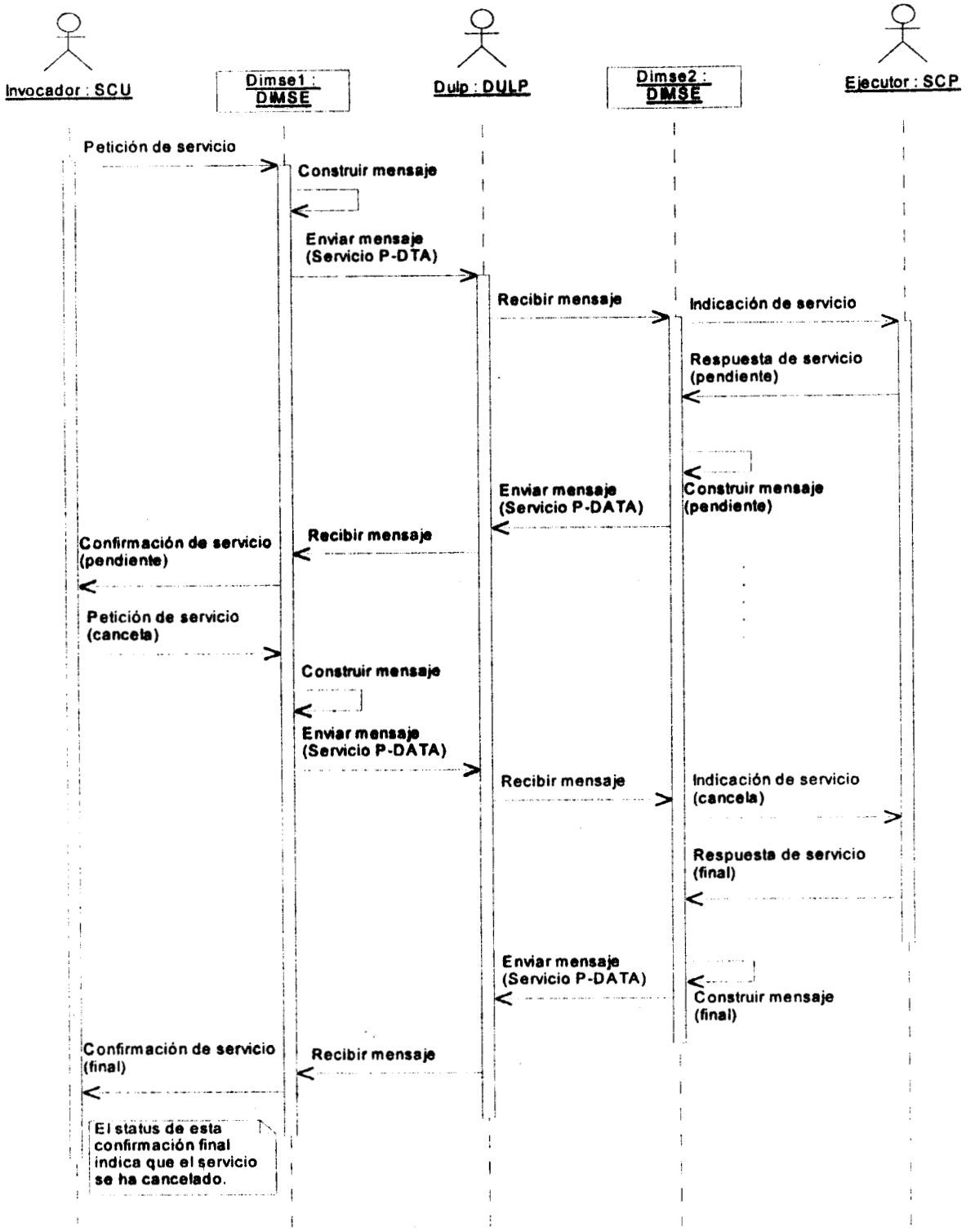

Figura 4.5 Diagrama de secuencia para el caso de uso Cancelar Respuesta Pendiente

Contratos

Los contratos ayudan a definir el comportamiento del sistema, describiendo el efecto de las operaciones sobre él. Su creación ocurre dentro de la fase de análisis y es dependiente del desarrollo anterior del modelo conceptual, diagramas de secuencia del sistema, y la identificación de las operaciones del sistema.

Los diagramas de secuencia mostrados anteriormente muestran los eventos del sistema generados por los actores externos, pero no muestran los detalles de la funcionalidad asociada con las operaciones del sistema, para subsanar esta deficiencia, los contratos, como se mencionó anteriormente, describen lo que una operación debe lograr. A continuación se describen los contratos para las operaciones de DIMSE identificadas en los diagramas de secuencia.

Nombre	Petición de servicio
Responsabilidades	Solicitar a DIMSE alguno de sus servicios para iniciar un intercambio de mensajes entre entidades de aplicación. Validar si la petición fue de alguno de los servicios soportados. Validar los parámetros necesarios para la prestación del servicio.
Pre – condiciones	Debe estar establecida la asociación entre dos entidades de aplicación.
Post – condiciones	Si el servicio fue válido: • Dimse.valido fue puesto a verdadero. • Se creó una instancia de Mensaje. • El Mensaje se asocio con DIMSE Si los parámetros son válidos: • Mensaje.valido fue puesto a verdadero En caso contrario: • Mensaje.valido fue puesto a falso En caso contrario: • Dimse.valido fue puesto a falso.

Nombre	Construir mensaje
Responsabilidades	Construir el mensaje DICOM con el Grupo de Comandos y el Grupo de Datos. Identificar si el servicio es una petición o respuesta Codificar el grupo de comandos. Etiquetar cada parámetro. Calcular la longitud del valor de cada parámetro y la del Grupo de Comando.
Pre – condiciones	Servicio y parámetros válidos
Post - condiciones	• Se creo un Grupo de Comando. • El Grupo de Comando nuevo se asocio con Mensaje

Nombre	Enviar mensaje
Responsabilidades	Mandar el mensaje DICOM utilizando el servicio P-DATA de Dulp. Fragmentar el mensaje en Grupo de Comandos y Grupo de Datos. Construir los PDV's necesarios hasta enviar el mensaje completo.
Pre – condiciones	Mensaje construido y validado.
Post - condiciones	• Creó una instancia de PDV. • El PDV se asoció con DIMSE. • Creó una instancia de Dulp. • Asoció a Dulp con DIMSE.

Nombre	Recibir mensaje
Responsabilidades	Recibir el mensaje DICOM enviado por el protocolo Dulp. Validar los PDV's que se reciben. Decodificar el Grupo de Comando. Dejar el Mensaje en el formato original para la EA.
Pre – condiciones	Existe una asociación entre DIMSE y DULP
Post – condiciones	• Se creó un PDV. (tal vez esto no es necesario?) • El PDV fue asociado con DIMSE. Si los PDV's son validos: • Pdv.valido fue puesto a verdadero. • Se creó un Mensaje. • El Mensaje fue asociado con DIMSE. • Se creó un Grupo de Comando. • El Grupo de Comando se asoció con Mensaje. En caso contrario: • Pdv.valido fue puesto a falso.

Nombre	Indicación de servicio
Responsabilidades	Informar a una Entidad de Aplicación (ejecutor) sobre la ejecución de una notificación u operación. Validar el servicio que se recibió. Emitir una primitiva de indicación para el ejecutor.
Pre – condiciones	DIMSE recibió un mensaje.
Post – condiciones	• Se destruyó la instancia Mensaje. Si el servicio fue válido: • Dimse.valido fue puesto a verdadero. • El sistema DIMSE quedó preparado para recibir una respuesta correspondiente a una notificación u operación. En caso contrario: • Dimse.valido fue puesto a falso.

Nombre	Respuesta de servicio
Responsabilidades	Responder al proveedor de servicio DIMSE sobre el resultado de la ejecución de una notificación u operación. Este resultado puede ser parcial, es decir, la notificación u operación esta en proceso, o bien definitivo, significando que la ejecución se ha concluido. Validar si la petición fue de alguno de los servicios soportados. Validar los parámetros necesarios para la prestación del servicio.
Pre – condiciones	El ejecutor recibió una indicación de servicio a través de DIMSE.
Post – condiciones	Si el servicio fue válido: • Dimse.valido fue puesto a verdadero. • Se creó una instancia de Mensaje. • El Mensaje se asocio con DIMSE. Si los parámetros son válidos: • Mensaje.valido fue puesto a verdadero. En caso contrario: • Mensaje.valido fue puesto a falso. En caso contrario: • Dimse.valido fue puesto a falso.

Nombre	Confirmación de servicio
Responsabilidades	Avisar a una Entidad de Aplicación (invocador) acerca de la respuesta sobre alguna petición de servicio previamente hecha. Validar el servicio que se recibió. Emitir una primitiva de confirmación al invocador.
Pre – condiciones	El invocador realizó una petición de servicio a DIMSE.
Post – condiciones	Si el servicio fue válido: • Dimse.valido fue puesto a verdadero. • Si la confirmación recibida fue parcial, el invocador sigue esperando otra confirmación. • Cuando la confirmación es concluyente (final) entonces se terminó el intercambio de mensaje. En caso contrario: • Dimse.valido fue puesto a falso.

4.2.2. Diseño

Durante un ciclo de desarrollo una vez terminada la fase de análisis se entra a la fase de diseño. Durante esta fase se realiza una solución lógica, cuya parte más importante es la creación de los diagramas de interacción, los cuales ilustran como se comunican los objetos para cumplir los requerimientos. Después de los diagramas de interacción, se dibuja el diseño de los diagramas de clase que resumen la definición de las clases que serán implementadas.

Diagramas de interacción

Los diagramas de colaboración son uno de los dos tipos de diagramas de interacción que muestran el orden de ejecución de los mensajes para realizar una operación. Los diagramas de colaboración muestran los objetos, sus relaciones y mensajes necesarios para llevar a cabo una operación.

La asignación de responsabilidades a los objetos participantes en una interacción, debe hacerse de manera tal, que se apliquen los principios fundamentales establecidos por los patrones GRASP. Los diagramas de colaboración son una buena herramienta para expresar esta asignación de responsabilidades y el patrón sobre el que se fundamenta dicha asignación.

Los diagramas de colaboración que se presentan enseguida muestran los objetos interactuantes para cada operación del sistema DIMSE, de los primeros se da una explicación y justificación de cada uno de los patrones en que se basa la asignación de responsabilidades de manera detallada con la finalidad de que se entienda como es que se usan los patrones GRASP. En los diagramas posteriores ya no se hace porque la justificación es la misma para cada patrón que se usa en la asignación de responsabilidades y sólo se muestra el patrón que la fundamenta.

El diagrama de colaboración de la figura 4.6 muestra la interacción de los diversos mensajes para la realización de la operación de petición de cualquiera de los servicios que proporciona DIMSE. La descripción detallada de cada uno de ellos es la siguiente:

1. Cuando un invocador desea empezar la transferencia de un mensaje, lo hace generando una petición de alguno de los servicios proporcionados por DIMSE, esta operación de petición la podemos considerar como un evento al cual debe responder el sistema, porque es una entrada externa al sistema originada por un actor externo, a saber, el invocador. De lo anterior, surge la pregunta: ¿Quién debe ser responsable de manejar un evento del sistema? Esta pregunta la responde el patrón Controlador, dado que él asigna la responsabilidad para manejar un mensaje de evento del sistema. Para nuestros propósitos, la clase DIMSE representará el sistema total, por lo tanto el controlador es del tipo fachada, y de ella se derivarán objetos de interfase responsables de manejar los eventos del sistema. La elección del controlador

fachada también se apoya en el hecho de que el número de operaciones del sistema son pocas (básicamente dos: petición y respuesta).

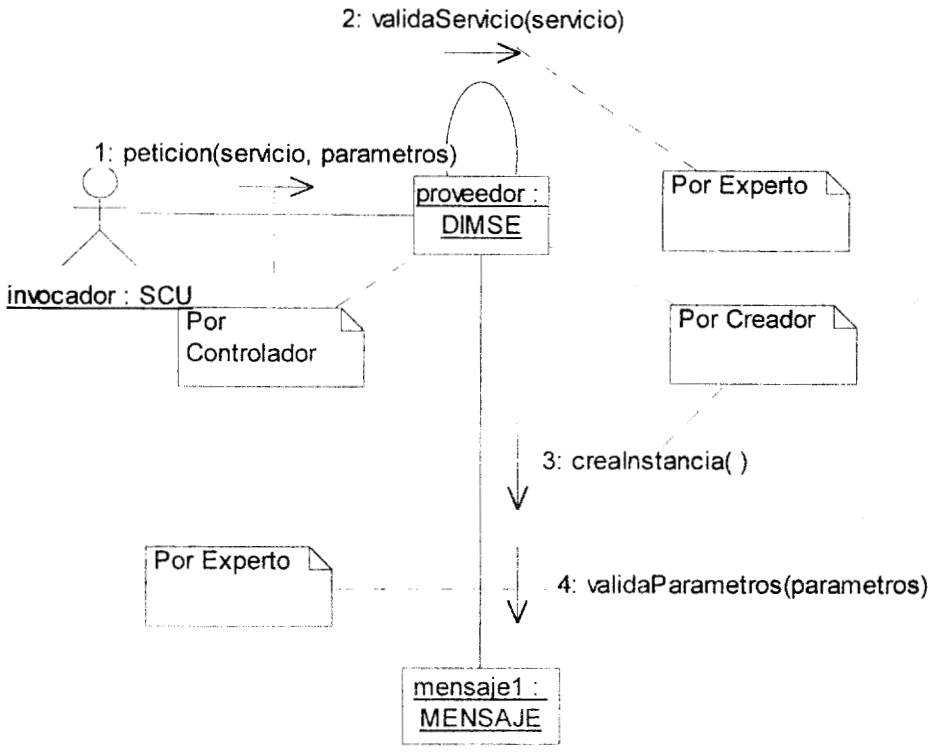

Figura 4.6 Diagrama de colaboración de la operación: peticion()

2. La operación de validación del servicio solicitado se le asigna al objeto proveedor de la clase DIMSE, debido a que él conoce cuales son los servicios que puede proporcionar, tiene la información necesaria para cumplir con la responsabilidad, es el experto de información. De aquí, que el patrón Experto apoye la asignación de esta responsabilidad.

3. Para poder validar los parámetros se debe crear primero la instancia del objeto que debe hacer la validación. ¿Quién será responsable de crear una instancia nueva? El patrón Creador asigna responsabilidades relacionadas con la creación de objetos que agregan, contienen, registran, usan estrechamente o tienen los datos inicializados del objeto creado. DIMSE es una buena opción para crear al objeto mensaje, dado que registra o usa estrechamente los mensajes ha intercambiar, de esta manera DIMSE tendrá una referencia al mensaje.

4. La validación de los parámetros para un servicio dado, requiere que el conocedor de lo que se necesita para construir un mensaje, certifique que el servicio venga acompañado de la información obligada para proporcionar dicho

servicio. Nuevamente, el patrón Experto apoya la asignación de esta responsabilidad a la clase MENSAJE, ya que ella tiene la información necesaria para determinar qué es lo que debe contener un mensaje.

El diagrama de colaboración para la operación de construir un mensaje, se muestra en la figura 4.7. Esta operación debe cumplir con la responsabilidad de codificar el mensaje con un grupo de comandos y un grupo de datos.

Este diagrama utiliza sólo dos patrones para la asignación de responsabilidades, el Creador y el Experto. El primero crea las diferentes instancias que son requeridas por la clase MENSAJE para construir el mensaje que se va a intercambiar, además de existir una relación de agregación ya que los comandos y datos se deben agregar al mensaje, así como los valores a los datos. Por otro lado, Experto sugiere la asignación de las otras responsabilidades porque cada objeto sabe como cumplir esa responsabilidad. La manera en la que se han asignado las responsabilidades, busca también un Bajo Acoplamiento para reducir el impacto que pueda tener el hacer cambios en alguna de las clases y que éstas sean más independientes.

Para que DIMSE cumpla con la responsabilidad de enviar un mensaje, los objetos de la figura 4.8 deben de relacionarse como lo muestra ese diagrama

Los patrones que ayudan en la asignación de responsabilidades para esta operación son: el Creador para crear la instancia de pdv1, y el Experto, que asigna a los objetos que conocen la información necesaria para realizar las tareas, la responsabilidad de ejecutarlas. Entre proveedor y pdv1 se cumple la opción de que el objeto proveedor contiene al objeto pdv1, lo que confirma el uso del patrón Creador

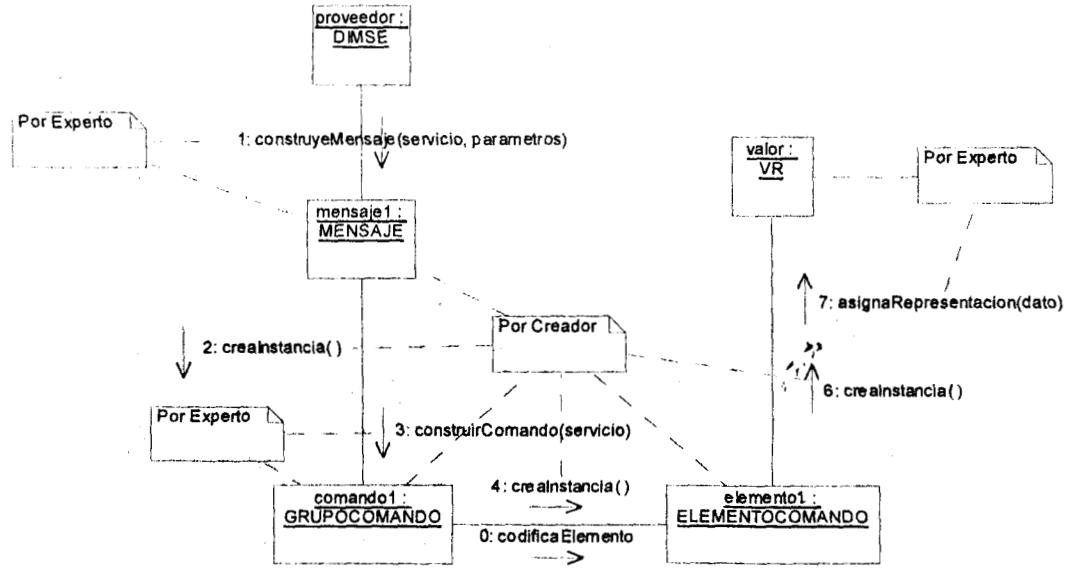

Figura 4.7 Diagrama de colaboración de la operación construyeMensaje()

DIMSE

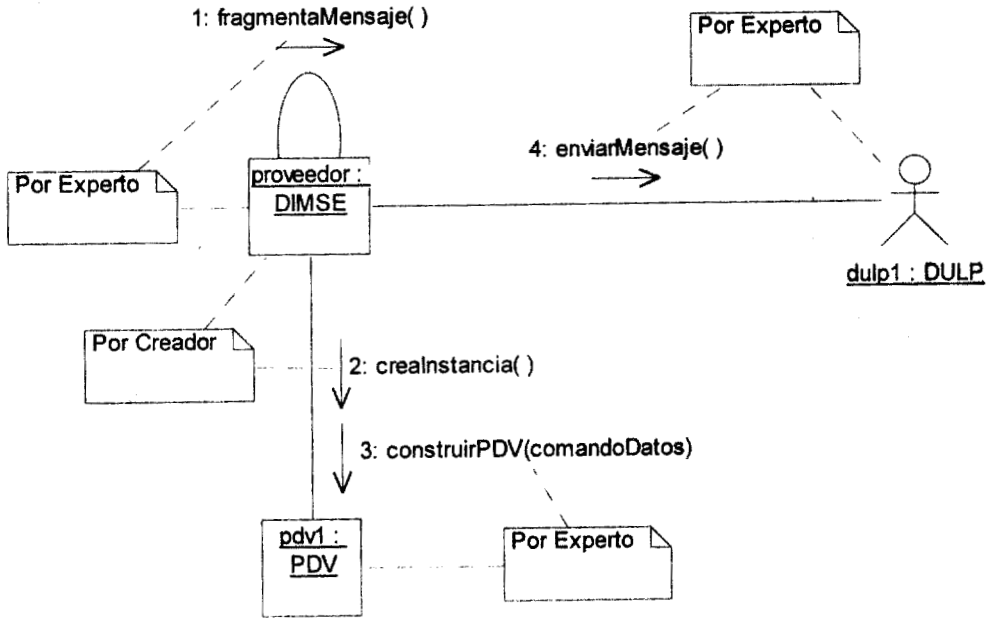

Figura 4.8 Diagrama de colaboración de la operación enviarMensaje()

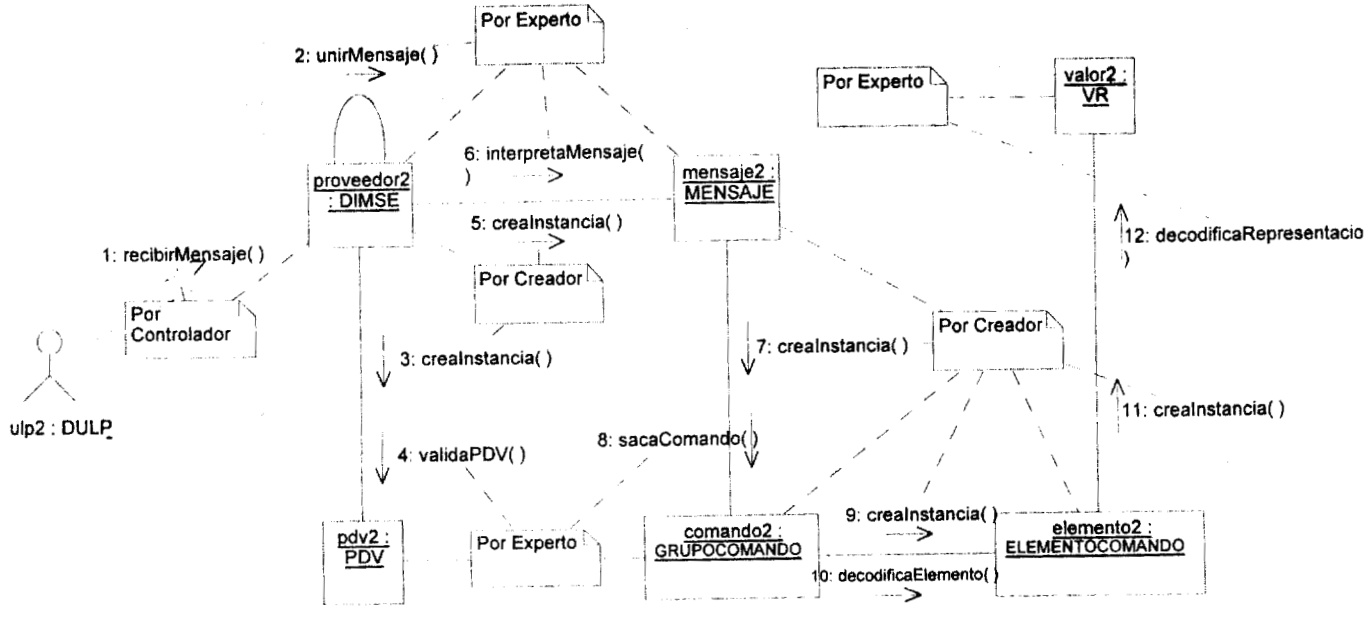

Figura 4.9 Diagrama de Colaboración de la operación recibirMensaje()

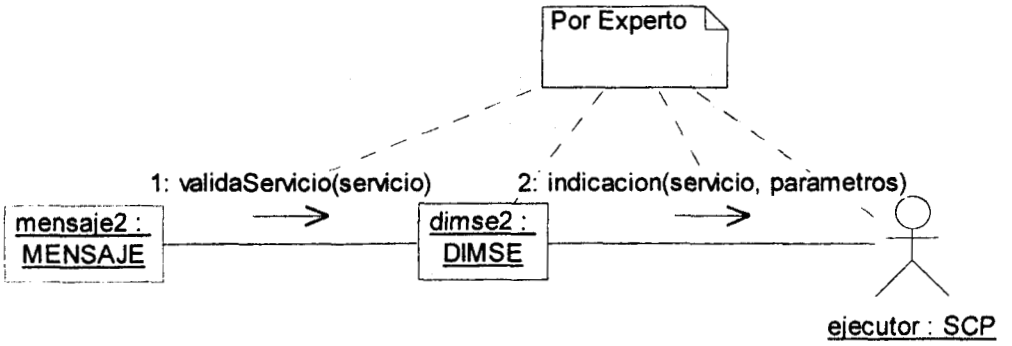

Figura 4.10 Diagrama de colaboración de la operación indicación()

La operación para que DIMSE reciba un mensaje es la más compleja de las operaciones, como puede apreciarse en su diagrama de colaboración de la figura 4.9. Aun así, los patrones básicos para asignación de responsabilidades son suficientes para modelar la operación completa. En este diagrama, nuevamente aparece el patrón Controlador asignando la responsabilidad de manejar un evento, la recepción de un mensaje, a DIMSE como representante del sistema completo. Las otras responsabilidades son asignadas por los patrones Creador y Experto, cuyo principio para asignar responsabilidades ya fue explicado.

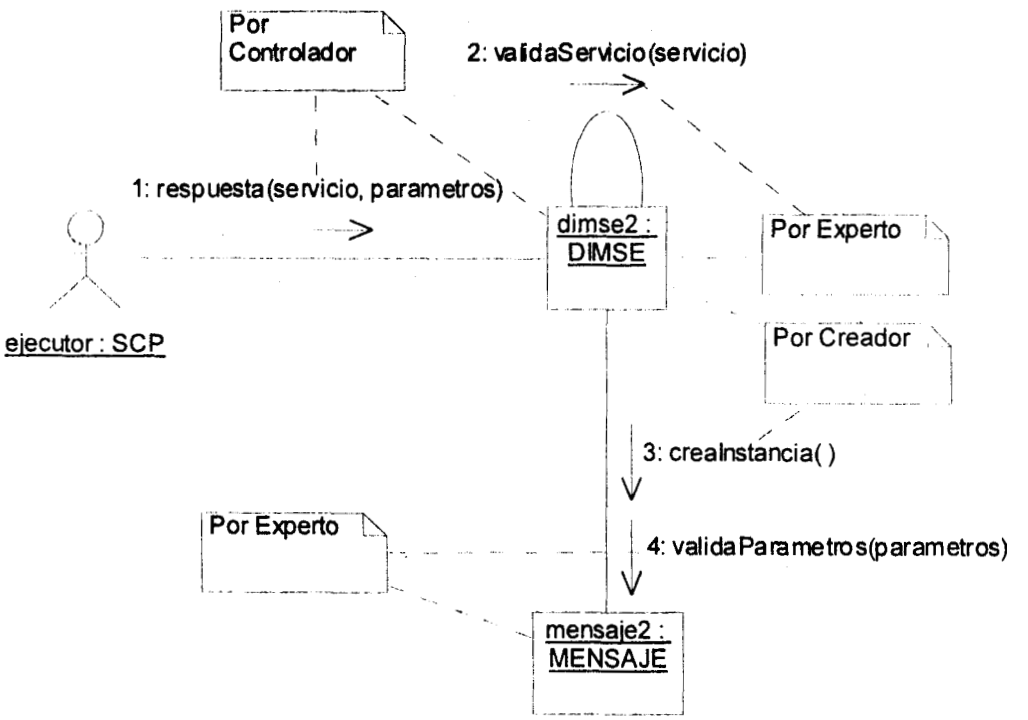

Figura 4.11 Diagrama de colaboración de la operación respuesta de servicio

Las figuras 4.10, 4.11 y 4.12 muestran los diagramas de colaboración faltantes para el modelado de todas las operaciones que requiere DIMSE para proporcionar todos los servicios que debe brindar este sistema.

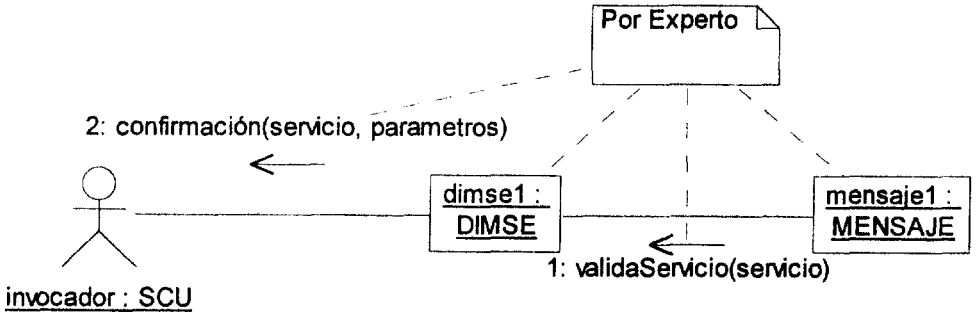

Figura 4.12 Diagrama de colaboración de la operación confirmación de servicio

Los diagramas de colaboración anteriores muestran las relaciones entre objetos y mensajes para realizar cada una de las operaciones necesarias para el protocolo DIMSE. Para aclarar y ejemplificar cómo estas operaciones en conjunto permiten el intercambio de un mensaje, las figuras 4.13 y 4.14, a través de un diagrama de secuencia, muestran la participación y el orden de uso de cada operación para modelar el proceso completo del intercambio de mensaje para el caso de uso Proveer Servicios DIMSE.

La figura 4.13 muestra las operaciones que se utilizan del lado del invocador de un servicio DIMSE. El invocador empieza creando un objeto proveedor de la clase DIMSE, para que posteriormente pueda solicitar alguno de los servicios proporcionados por DIMSE. La solicitud se inicia cuando el invocador hace una llamada a la operación peticion() del proveedor DIMSE, esta operación lleva como parámetro un buffer de datos, que contiene el servicio solicitado y los parámetros requeridos por tal servicio. Una vez recibida la petición del servicio, se validan tanto el servicio solicitado como los parámetros necesarios por los objetos proveedor y mensaje1 respectivamente. En caso de que alguna de estas validaciones falle no será posible otorgar el servicio solicitado. El diagrama de la figura 4.13 no muestra el manejo de esta excepción, sin embargo, la manera de notificarlo al invocador es a través de la emisión de una respuesta temprana como lo establece el estándar.

Después de las validaciones se construye el mensaje, creando el Conjunto de Comandos agregándole cada uno de sus elementos con su respectiva representación de los datos. Posteriormente, dependiendo de la longitud del mensaje, se fragmenta en PDV's para ser enviados utilizando la operación enviarMensaje(), la cual no es otra cosa que el servicio P-DATA de la interfase DULP.

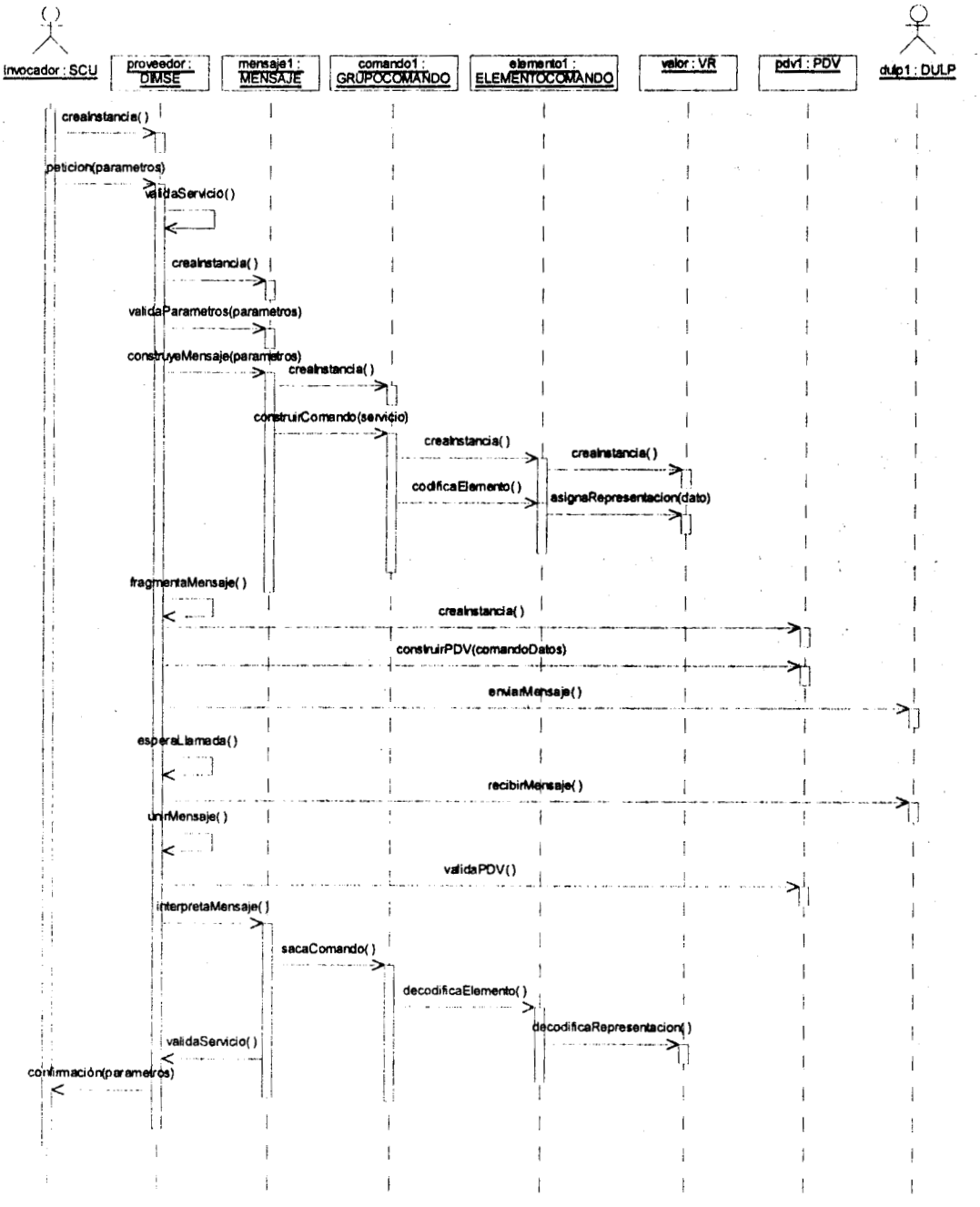

Figura 4.13 Diagrama de secuencia de un servicio DIMSE del lado del invocador

Una vez enviado el mensaje el protocolo DIMSE queda en un estado de espera, por medio de la operación esperarLlamada(), verificando si llega una petición de la entidad de aplicación o una respuesta de la interfase Dulp, como todos los servicios son confirmados, la figura 4.13 muestra la llegada de la respuesta que se esta esperando para la petición del servicio solicitado. Al llegar esta respuesta la operación recibirMensaje() se encarga de recibirla, iniciándose una etapa de proceso inversa a la que se realizó al recibir la petición, es decir, si el mensaje viene fragmentado, la operación unirMensaje() se encarga de unir el mensaje en una sola entidad, validando cada PDV que se recibe.

Al tener completo el mensaje se inicia la decodificación del mismo por medio de la operación interpretaMensaje(), la cual se apoya en sus colaboradores: sacaComando(), decodificaElemento() y decodificaRepresetacion() para realizar esta tarea, con la finalidad de poder entregar el mensaje en el formato que requiere la entidad de aplicación. Terminada la decodificación del mensaje, se valida la respuesta del servicio y se pasa esta respuesta a la entidad que solicitó el servicio por medio de la operación confirmación(), la cual conlleva el resultado de la operación solicitada al ejecutor.

La figura 4.14 muestra las operaciones que se realizan del lado de la entidad de aplicación (ejecutor) que ejecuta una operación solicitada. En este caso, el diagrama de secuencia empieza con la creación de una instancia de DIMSE por el ejecutor para que se este preparado para atender solicitudes, ya sean de la entidad de aplicación o de las capas inferiores a través de la interfase Dulp. La figura 4.14 muestra la llegada de un mensaje desde la interfase Dulp, en ella se observa que las operaciones que intervienen en el proceso de recepción son las mismas que las mostradas en la figura 4.13, aplicadas en el orden necesario para informar al ejecutor, por medio de la operación indicacion(), del servicio solicitado.

Después de pasar la indicación del servicio al ejecutor, la instancia de DIMSE (proveedor) se queda en un estado de espera. Terminada la ejecución de la operación por parte del ejecutor, éste envía una respuesta del resultado de la misma a través de la operación respuesta(), la cual recibe un tratamiento análogo a la operación petición() de la figura 4.13.

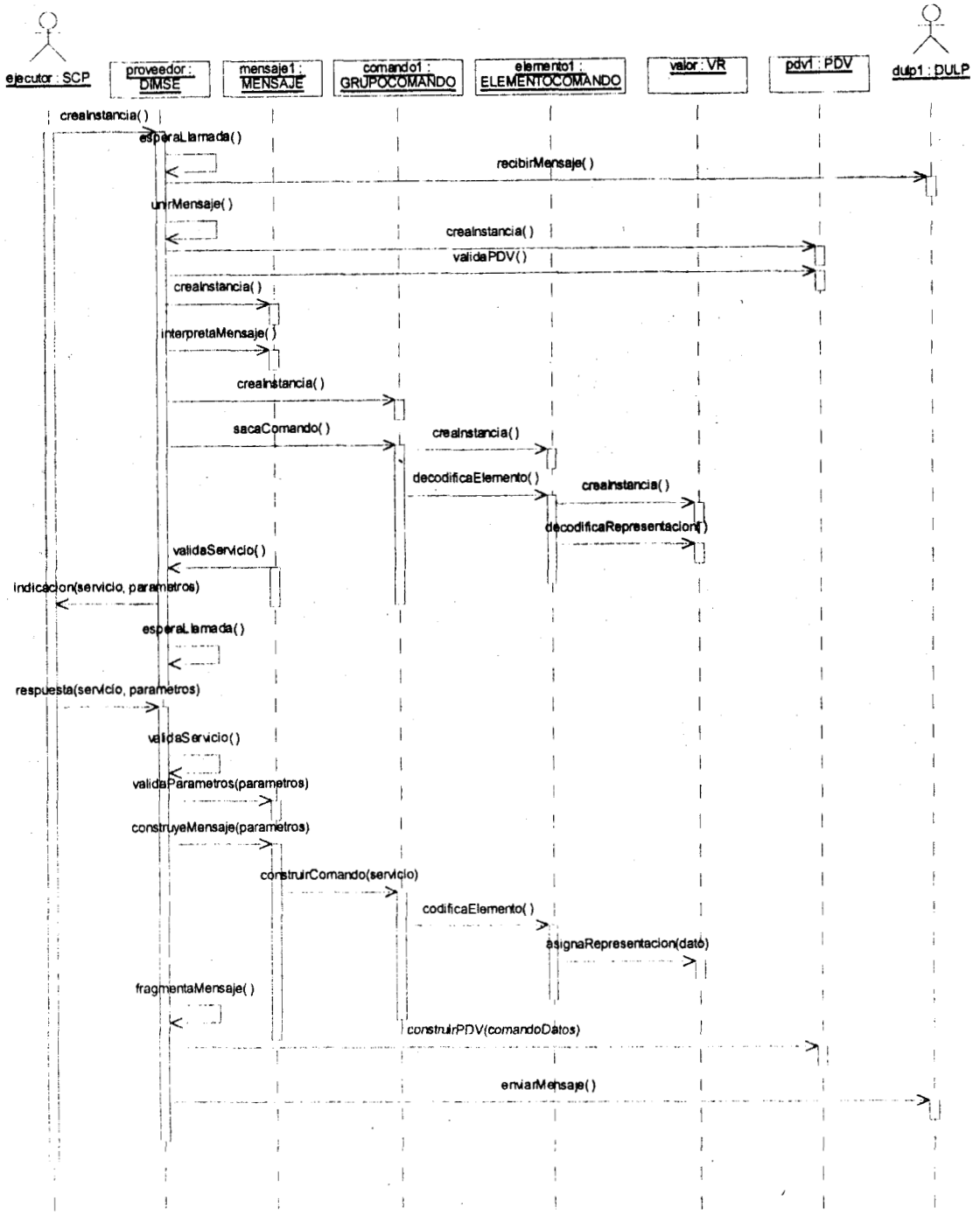

Figura 4.14 Diagrama de secuencia de un servicio DIMSE del lado del ejecutor

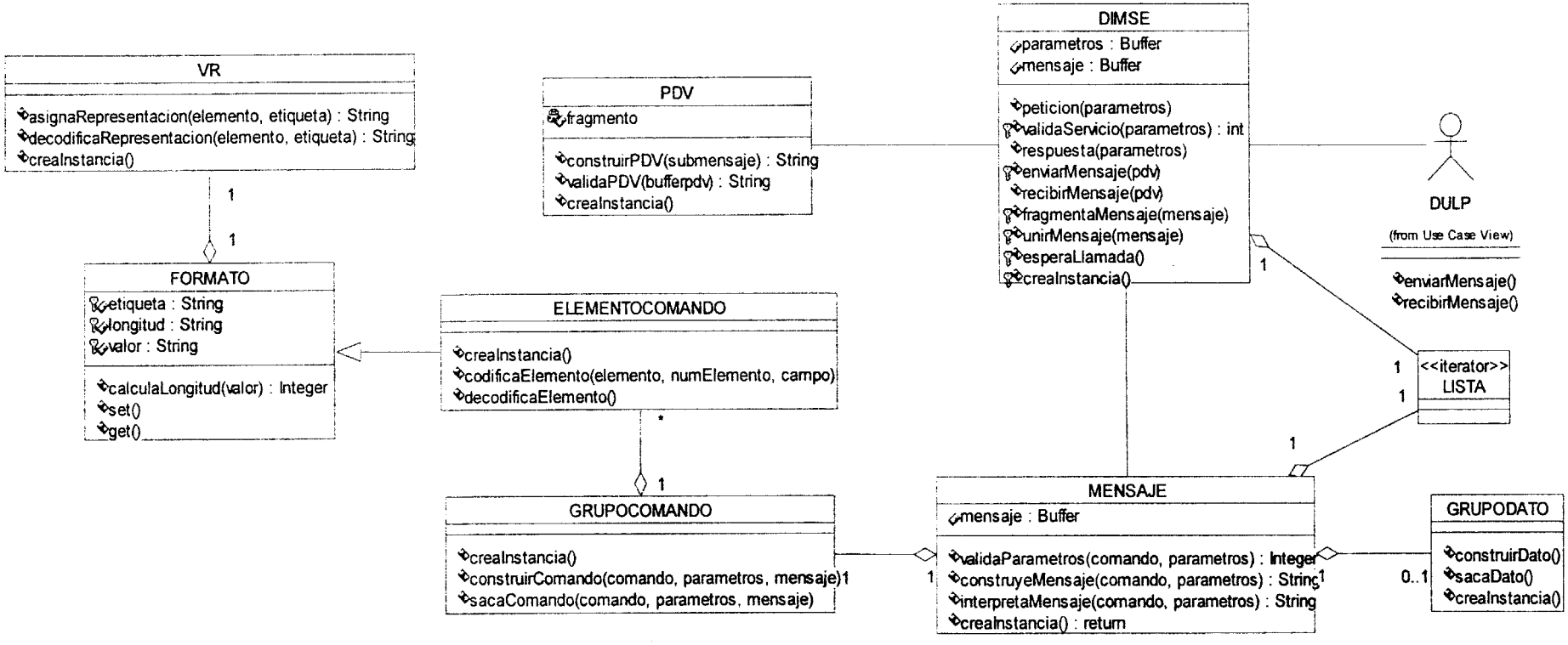

Figura 4.15 Diagrama de clases de DIMSE

Diagrama de clases

Un diagrama de clases muestra las definiciones para las entidades de software en lugar de los conceptos del mundo real, es por eso que en su creación se deben identificar aquellas clases que participan en la solución de software. Estas clases se pueden encontrar examinando el modelo conceptual y los diagramas de interacción. Realizando un análisis de los diagramas de interacción se obtiene el diagrama de clases de la figura 4.15.

De este diagrama se observa que la clase principal se llama DIMSE, representando al sistema total pero ahora como una entidad de software. Las clases restantes son necesarias como colaboradoras para cumplir con la responsabilidad principal de intercambiar mensajes entre Entidades de Aplicación. Por ejemplo, cuando DIMSE recibe la petición de alguno de sus servicios por parte de un usuario, primeramente valida que el servicio solicitado sea de los que ella puede proporcionar, una vez validado, delega la responsabilidad de construir el mensaje a la clase Mensaje, la cual a su vez, se apoya en las clases COMANDO y DATO para agregar los comandos y datos correspondientes al mensaje por medio de una relación de agregación. Ya construido el mensaje, la clase DIMSE usa como colaborador a la clase PDV para que fragmente el mensaje, y así poder enviarlo utilizando a DULP, que es el Protocolo de Capa Superior encargado de comunicarse con las capas inferiores de TCP/IP para realizar la transferencia del mensaje.

4.2.3. Construir

La fase de construcción involucra la implementación del diseño en software y hardware. Las actividades que se realizaron en esta fase fueron la definición (implementación) de las clases, la definición de los métodos y la escritura de código de prueba.

Como se mencionó anteriormente del diseño del diagrama de clases se obtiene como mínimo el nombre de la clase, los métodos signados y atributos de la clase, siendo esto suficiente para crear la definición de una clase.

La figura 4.16 es una parte del diagrama de clases mostrando la clase DIMSE y la relación que guarda con la clase MENSAJE. La traducción a código de C++ de la definición de la clase DIMSE es prácticamente directa como se muestra a continuación:

```
class Dimse
{
public:
        Dimse();
        ~Dimse();
        void peticion(Buffer* parametros);
        void recibirMensaje(char* pdv);
```

```
protected:

private:
        Buffer* mensaje
        int validaServicio(Buffer* parametros);
        void enviarMensaje(char* pdv);
        void fragmentaMensaje(Buffer* mensaje);
        void unirMensaje(Buffer* mensaje);
        void esperaLlamanda(void);
};
```

El método crealnstancia() no esta definido explícitamente en la clase DIMSE, dado que las responsabilidades de este método las realiza el constructor de la clase, es decir, Dimse().

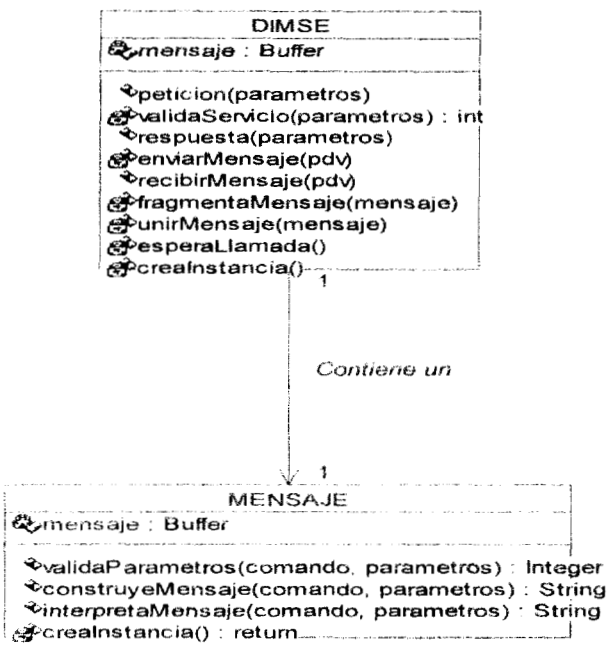

Figura 4.16 La clase DIMSE y su relación con la clase MENSAJE

Otro método que tampoco se codifica en la implementación de la clase es respuesta(), lo anterior obedece a que cuando el ejecutor recibe una indicación de servicio DIMSE, también recibe el control del programa y se lo regresará a DIMSE cuando termine de realizar el servicio solicitado. Una vez que el ejecutor termina el servicio, le devuelve el control del programa a DIMSE y ésto se interpreta como una primitiva de respuesta implícita. El método respuesta() se representa explícitamente en los diagramas de interacción y de clases para agregar claridad a los mismos.

La clase DIMSE tiene una asociación con MENSAJE y con navegabilidad hacia ella. Es común interpretar esto como un atributo de referencia en la clase DIMSE, declarando una instancia de MENSAJE como un atributo de DIMSE. La definición de DIMSE se modifica agregando la siguiente línea en la parte privada de la clase:

```
Mensaje* mensaje1;
```

Aunque se agrega una instancia de MENSAJE a DIMSE, ésta no se declara explícitamente en la sección de atributos de la clase, porque existe una visibilidad de atributo de referencia sugerida por la asociación y navegabilidad.

En el segmento del diagrama de clase de la figura 4.16 no están presentes los nombres de los roles en ninguno de los extremos de la asociación, por tal motivo hemos llamado arbitrariamente la instancia de MENSAJE como mensaje1 en la clase DIMSE.

Para ilustrar la creación de los métodos a partir de su diagramas de colaboración usaremos el correspondiente a peticion(), el cual se encuentra definido en su clase DIMSE. El mensaje peticion() es enviado por un SCU (invocador) a DIMSE para solicitar alguno de los servicios proporcionados por este último, razón por la cual el método esta definido en la parte pública de DIMSE.

Observando el diagrama de colaboración de la figura 4.6, se nota que como respuesta a la recepción del mensaje peticion(), el proveedor DIMSE debe validar el servicio que le están solicitando por medio de la operación validaServicio(). Este método debe regresar el nombre del servicio solicitado, si es que la petición() es valida. El segundo mensaje que se observa en este diagrama es el que se manda a MENSAJE, para que valide a su vez si los parámetros asociados con el servicio solicitado, son los necesarios para proporcionar dicho servicio. Este segundo mensaje será enviado siempre y cuando se haya solicitado un servicio valido. En caso de que alguna o ambas validaciones sean falsas, se debe emitir una respuesta temprana para el invocador. El código en C++ que produce el diagrama de colaboración para el método peticion() es el siguiente:

```
void Dimse::peticion(Buffer* parametros)
{
        enum servicio ServicioDimse;
        enum Dimse_C Comando;

        if(ServicioDimse = validaServicio(parametros))
                if(mensaje1->validaParametros(Comando,parametros))
        else
                respuestaTemprana(parametros);
        else
                respuestaTemprana(parametros);
}
```

Los dos ejemplos anteriores de definición de clase y de método, muestran como el proceso da la pauta para la implementación en software de las dos definiciones básicas que deben realizarce cuando se escribe código fuente en el desarrollo de un sistema. En los resultados solo se ha incluido un ejemplo de cada definición para ilustrar la aplicación del proceso de desarrollo en esta fase. Para las demás clases y sus métodos se aplica de la misma manera, considerando todos los cambios que surjan durante esta fase, a pesar de que en los ejemplos anteriores el proceso de traducción es relativamente mecánico. Sin embargo, durante la codificación se tienen que resolver todos los detalles no cubiertos por las fases anteriores y realizar todos los cambios pertinentes, aplicando el proceso de manera iterativa e incremental desde la fase que sea necesaria para resolver estos detalles.

5. Discusión

5.1. DICOM

DICOM, Imagen Digital y Comunicaciones en Medicina, es el estándar de la industria para transferir imágenes médicas e información entre dispositivos electrónicos. Su meta es una arquitectura abierta que permita a sus usuarios integrar equipo de imágenes de distintos fabricantes y que soporte un rango amplio de modalidades.

El estándar DICOM no solo es una definición de un formato para intercambiar imágenes médicas, sino que define las estructuras de datos para las imágenes médicas y sus datos relacionados, los servicios de red, los formatos de almacenamiento y los requerimientos de conformidad de los dispositivos y programas.

El estándar DICOM esta dividido en varias partes, cada una describiendo un tema principal como las Clases de Servicio, las Definiciones de Objeto de Información (IOD), el Intercambio de Mensajes (DIMSE), etc. Esta división del estándar dificulta la comprensión del mismo porque cada tema se aborda en forma independiente de los demás, y para un novato en el tema es muy fácil perderse en un sin número de conceptos y abreviaciones que corresponden a otras partes del estándar. Aunado a lo anterior, la definición del estándar es muy escueta en cuanto a la información que ofrece, y esta dirigido a los expertos del tema. El tratar de entenderlo es una tarea ardua que consume mucho tiempo, sobre todo si se intenta hacerlo en forma individual o con un equipo muy reducido, como fue nuestro caso. Para hacerle frente al estándar, y sobre todo si se pretende hacer una implementación del mismo, se requiere de un grupo interdisciplinario, con experiencia en los diversos aspectos que cubre como son: modalidades de imágenes, modelo orientado a objetos, protocolos de red, servicios de red, etc.

5.2. DIMSE

El Elemento de Servicio de Mensaje DICOM, DIMSE, incide dentro de los servicios de red del estándar. Estos servicios de red se basan en el modelo cliente-servidor, razón por la cual, antes de que dos entidades de aplicación puedan intercambiar información, se debe establecer una conexión y negociar una asociación entre ellas para acordar quién jugara el papel de cliente y quién el papel de servidor, cuales servicios serán soportados y en que formato se transmitirán los datos.

En el dominio de la aplicación la relación entre el cliente y el servidor esta definida por la Clase de Servicio y en el dominio de intercambio por la Sintaxis de Transferencia. La Clase de Servicio contiene la descripción de la información que

maneja y de las operaciones que se pueden realizar sobre esta información. Usando el modelo de objetos, la Clase de Servicio, la información y las operaciones se combinan para dar origen a la definición de la clase, llamada una Clase Par Objeto Servicio o Clase SOP.

Los servicios proporcionados por el dominio de intercambio son las operaciones de la Clase de Servicio, y son estos servicios lo que permite que el cliente y el servidor puedan funcionar en un ambiente de proceso distribuido.

El dominio de intercambio es el dominio de DIMSE, razón por la que tiene la responsabilidad de proporcionar los servicios necesarios para realizar el intercambio de un mensaje. DIMSE proporciona básicamente dos tipos de servicio: los compuestos, conocidos como DIMSE-C y los normalizados, conocidos como DIMSE-N.

Los servicios normalizados son usados por las Clases de Servicio que utilizan IODs normalizados, es decir, aquellos que consisten de una sola entidad de información. Cada entidad contiene información de una sola cosa del mundo real como es un paciente, una imagen, etc. Las clases que usan los servicios normalizados realizan funciones de administración.

Las Clases SOP compuestas utilizan los servicios compuestos de DIMSE, dado que contienen una combinación de entidades de información, llamada IOD compuesto. Estas clases manipulan el flujo de los datos de la imagen, siendo estos una estructura de información compleja.

Antes de que se lleve a cabo el intercambio de datos de una instancia SOP a través de la red, se determina la manera en que los datos se codifican en una sarta ("stream") de bytes. La manera de realizar esta codificación se especifica en la Sintaxis de Transferencia. Los tres aspectos que se tienen que definir en la sintaxis de transferencia son:

1. Como representar un valor: asignando un VR (Valor de Representación) para cada tipo de dato.
2. El orden de bytes para números con bytes múltiples: "little endian" o "big endian"
3. En caso de compresión: el formato de la compresión.

5.3. Proceso de desarrollo

El proceso que se aplicó en el desarrollo del sistema DIMSE es el propuesto por Larman, y como él mismo dice: "es un proceso de desarrollo, describiendo un posible orden de actividades y un ciclo de vida de desarrollo. Sin embargo, no es un método o proceso definitivo, proporciona un ejemplo de pasos comunes" [Larman98] que son aplicables en un proyecto de software en una variedad amplia de dominios de problema.

Aplicar un proceso de desarrollo a un proyecto de software y realizar un análisis y diseño orientado a objetos toma tiempo, y conlleva un incremento en el costo del proyecto, sobre todo si los participantes no tienen experiencia en el uso de este tipo de metodologías. Una gran parte, si no es que la mayoría, de los profesionistas que se dedican al desarrollo de software empiezan inmediatamente con la programación, dejando de lado el análisis y el diseño. ¿Cuáles son las razones que hacen conveniente la aplicación de un proceso de desarrollo y el análisis y diseño orientado a objetos?, algunas de esta razones son:

- Los proyectos de software son un negocio de riesgo, muchos proyectos nunca se concluyen, muchos son más caros de lo que se estimó en un principio y otros son cancelados; así es de que una de las razones principales es para disminuir el riesgo e incrementar la probabilidad de tener un proyecto exitoso.

- Es útil crear procesos que se puedan repetir tanto por individuos como por equipos. El desarrollo que confía en el esfuerzo individual no es sustentable.

- Es más económico y fácil realizar cambios durante las actividades del análisis y diseño que durante la fase de construcción.

- El modelado de sistemas y la abstracción para encontrar los detalles esenciales descomponen y facilitan el manejo de la complejidad.

- La creación de sistemas robustos, mantenibles y con incremento de soporte de reuso de software, se logran si se aplica una metodología de diseño antes de programar.

En el presente trabajo se ha considerado conveniente aplicar un proceso de desarrollo, describiendo un posible orden de actividades y un ciclo de desarrollo. Sin embargo, no es un método o proceso definitivo, proporciona un ejemplo de pasos comunes de desarrollo y las técnicas del análisis y diseño orientado a objetos con la finalidad de lograr las ventajas arriba mencionadas, no se pretende ser un experto en la aplicación de la metodología, porque no es el objetivo de la Ingeniería Biomédica, eso se le deja a los profesionales de la computación, sin embargo, no hemos querido seguir cometiendo el error, en el que incurren muchas organizaciones, que al querer cuidar los centavos descuidan los pesos cuando optan por no usar las técnicas formales del análisis y diseño, y deciden entrar de manera directa y precipitada a la codificación.

Los diferentes productos obtenidos en cada una de las fases de este proceso de desarrollo, como son el modelo conceptual, casos de uso, diagramas de secuencia, etc., han sido el resultado final de aplicar en forma iterativa e incremental el proceso de desarrollo.

Como ya se mencionó, los productos que se pueden obtener o las actividades que se deben realizar en cada fase del proceso de desarrollo son muy variadas,

consideramos que el proceso de desarrollo no se debe aplicar de manera inflexible y al pie de la letra, sino que este debe adecuarse a las necesidades y requerimientos del dominio del problema en cuestión, realizando sólo aquellas actividades que ayuden a descomponer la complejidad del mismo y que aporten comprensión y claridad para establecer el modelo de solución adecuado. El proceso de desarrollo no es una receta mágica para producir sistemas de software exitosos, pero si da la pauta y dirección a seguir para lograr el éxito del proyecto, si los participantes en el mismo tienen la habilidad y dominio suficiente del proceso de desarrollo. Con esto, lo que quiero decir, es que los participantes en el proyecto deben comprender muy bien las fases del proceso, y desarrollar la habilidad, creatividad e inventiva para aplicarlos adecuadamente con la finalidad de crear buenos diseños y con esto aumentar la probabilidad de alcanzar el éxito del proyecto.

[Larman98] y [Jacobson99] coinciden en cuanto a que el proceso de desarrollo de proyectos de software debe ser: Manejado por casos de uso, centrado en la arquitectura e iterativo e incremental. Otra coincidencia que tienen es en cuanto al flujo de trabajo de [Jacobson99] y las actividades de [Larman98]: Requerimientos, análisis, diseño, implementación y prueba. La diferencia en los procesos que proponen se da en los pasos mayores o fases de alto nivel que debe tener el proceso, [Jacobson99] propone que sean cuatro: Concepción, Elaboración, Construcción y Transición; mientras [Larman98] propone: Planear y elaborar, Construir y Entregar. Sin embargo, la diferencia sólo es en cuanto a número y nombre ya que en la práctica los aspectos generales que cubren son los mismos. En general, se puede decir que ambos autores coinciden en las fases que debe presentar cada proyecto así como en las actividades que deben realizarse. La diferencia en los procesos esta en la manera muy particular de realizar cada una de las actividades, por ejemplo, [Larman98] utiliza los patrones GRASP para asignación de responsabilidades y [Jacobson99] lo hace sólo de manera enunciativa.

A pesar de estas coincidencias, una diferencia importante es que [Larman98] no desarrolla la última fase del proyecto, es decir, la Entrega y [Jacobson99] si le dedica un pequeño capítulo en su libro.

A continuación se mencionan las actividades que se realizaron, dentro de los pasos a nivel macro del proceso, los cuales son: Planear y Elaborar, Construir y Entregar (Deploy).

5.4. Planear y Elaborar

La fase de Planear y Elaborar incluye la concepción inicial, investigación de alternativas, planeación, especificación de requerimientos, etc. Los artefactos que se generaron son: los requerimientos y los casos de uso.

Requerimientos

Los requerimientos del sistema se definen enunciando de manera declarativa el propósito del sistema, de dónde surge su necesidad, sus metas, funciones y atributos del sistema. Al describirse las necesidades del sistema se identifican los alcances, fronteras y contexto del mismo, lográndose así tener una idea clara de lo que es el dominio del problema.

Casos de uso – Diagrama de casos de uso

Los casos de uso son las secuencias de eventos que un usuario del sistema (actor) usa para completar un proceso. Los casos de uso permiten identificar las diferentes maneras en que será usado el sistema, así como los procesos que debe ser capaz de realizar. Son una herramienta útil para expresar lo que se ha comprendido del dominio del problema y establece una forma de comunicación entre el desarrollador o equipo de desarrollo y los futuros usuarios del sistema; al ser presentado y explicado el conjunto de los casos de uso y la relación que existe entre ellos, por medio de un diagrama de casos de uso, a los usuarios y expertos del dominio se determina si se han comprendido las necesidades y procesos a satisfacer por el sistema.

Los casos de uso se describen de manera narrativa en un formato de alto nivel para expresar rápidamente lo que se ha comprendido del problema y la funcionalidad que debe lograr el sistema. Para agregar información más detallada del dominio del problema se utilizan los formatos extendidos de los casos de uso a fin de aclarar y mejorar el entendimiento de ellos.

La definición de requerimientos se definió de manera fácil ya que éstos se encuentran analizando la parte 7 del Estándar DICOM, sin embargo, no sucedió lo mismo con los casos de uso. Los ciclos de vida del proceso iterativos e incrementales permitieron depurar y refinar los casos de uso hasta lograr un diagrama de estos casos que permitió explicar a un alto nivel cada uno de los servicios que debe proporcionar DIMSE. El diagrama de casos de uso probó su utilidad en el proceso de desarrollo porque nos obligó a buscar y entender cada uno de los eventos a los que el sistema debe responder. Aprendimos que cuando el desarrollador es capaz de explicar la funcionalidad del sistema a través de los casos de uso al usuario o experto del dominio, y éste lo entiende y esta de acuerdo en que cumple con todos los procesos que debe proporcionar el sistema, entonces el diagrama de casos de uso esta terminado.

Consideramos que estas dos actividades son suficientes para comprender la complejidad y la funcionalidad que debe tener el Protocolo de Servicio de Intercambio de Mensajes DICOM, estando así preparados para comenzar el siguiente paso del proceso, el Construir.

5.5. Construir (Build)

La fase principal del proceso de desarrollo es la de Construir, teniendo como propósito desarrollar el sistema completo. Es en esta parte del proceso donde tienen lugar los ciclos de desarrollo, cuyo ciclo de vida iterativo esta basado en el refinamiento sucesivo del sistema a través de múltiples ciclos de análisis, diseño y construcción. Cada ciclo afronta un conjunto relativamente pequeño de los requerimientos y el sistema crece incrementalmente a medida que cada ciclo se completa.

Análisis

El análisis enfatiza la investigación del problema y no el cómo definir una solución. Desde la perspectiva de los objetos enfatiza la consideración del dominio del problema para encontrar y describir sus objetos (o conceptos).

La primera actividad dentro del análisis es la elaboración de un modelo conceptual que describa los conceptos (objetos) principales del dominio del problema. En el caso particular de DIMSE, la identificación de los conceptos y la manera en que se relacionan permite establecer, en un nivel alto de abstracción, la forma en la que se lleva a cabo el intercambio de mensajes. El modelo conceptual asienta las bases que en el futuro servirán para definir el diagrama de clases.

Una vez identificados los conceptos, la atención debe centrarse en el comportamiento del sistema. El proceso de desarrollo establece los diagramas de secuencia como la actividad para describir lo que hace el sistema sin explicar cómo se hace. Estos diagramas facilitan y ayudan en la compresión del comportamiento del sistema. Dado que lo que se trata de resolver en este dominio de problema es un protocolo, esta actividad fue importante y decisiva para entenderlo e identificar las operaciones del sistema.

Identificadas las operaciones del sistema, la siguiente actividad da como producto los contratos de las operaciones, los cuales describen lo que ellas deben lograr. Las partes más importantes de estos contratos son el establecimiento de las responsabilidades de cada operación y los efectos o cambios de estado que producen las operaciones en el sistema, es decir, las post-condiciones.

El modelo conceptual establece un esbozo de lo que en el futuro será el diagrama de clases. La identificación de los conceptos no presentó mucha dificultad, sin embargo, el modelo inicial se modificó debido a que no se había entendido correctamente el concepto de lo que es el Grupo de Comando. La característica del proceso de ser iterativo e incremental, permitió que en una etapa posterior se manifestará el error, obligándonos a regresar al modelo conceptual para modificarlo y corregir el problema.

Los diagramas de secuencia fueron una herramienta que nos fue de mucha utilidad para entender el comportamiento del sistema. Elaboramos muchos

diagramas de secuencia intentando con ello entender el comportamiento del sistema, fue a través de un refinamiento sucesivo de ellos y a través de repasar muchas veces la especificación del Estándar que se logró establecer los diagramas que modelaron de correctamente el sistema.

La etapa del análisis, en particular, fue difícil debido principalmente a los malos hábitos que se adquieren cuando se desarrolla un sistema sin aplicar una metodología en el proceso de elaboración del mismo. Nos ocasionó mucha pérdida de tiempo el querer responder el cómo, cuando la atención principal del análisis es el qué debe hacer el sistema. Sólo cuando pudimos concentrarnos en el qué y dejando de lado el cómo fue que pudimos avanzar en esta etapa.

El modelo conceptual, los diagramas de secuencia y los contratos de las operaciones fueron las actividades suficientes y necesarias para concluir la investigación del dominio del problema y determinar lo que debe hacer el sistema DIMSE.

Diseño

Para desarrollar una aplicación es necesaria la descripción detallada de la solución lógica y cómo ella satisface los requerimientos y restricciones, es por esto que el diseño enfatiza la definición de una solución lógica. En el modelo orientado a objetos el énfasis esta sobre la definición de los objetos de software que serán implementados en un lenguaje de programación orientado a objetos.

La parte esencial de esta fase del desarrollo es la creación de los diagramas de interacción, ilustrando con ellos la forma en que se relacionan los objetos para cumplir los requerimientos del sistema. Aunque UML define dos clases de diagramas de interacción: los de secuencia y los de colaboración, en el proceso se enfatiza el uso de los segundos debido a su excepcional expresividad, habilidad para dar a conocer información contextual, y su relativa economía espacial [Larman98].

En la creación de los diagramas de colaboración se utilizan a los patrones GRASP para la asignación de responsabilidades a los objetos, esta es una actividad ineludible y tiene un gran efecto sobre la robustez, mantenimiento y reusabilidad de los componentes de software. En la fase de diseño la etapa más importante es la de asignación de responsabilidades dentro de los diagramas de interacción, por lo que se le debe poner mucha atención no escatimando el tiempo que se le dedique a este paso. La asignación de responsabilidades con patrones GRASP es análogo al uso de las tarjetas CRC (Clase Responsabilidad Colaborador) en otros procesos de desarrollo.

El enfoque del uso de los patrones en este proceso de desarrollo es diferente, al que le da Gama en su libro [Gama95]. Gama menciona que sus patrones de diseño son "descripciones de objetos y clases comunicándose que están hechos a la medida para resolver un problema de diseño general en un contexto particular".

Gama explica cada uno de los patrones contenidos en su catálogo, llegando inclusive a ejemplificar la implementación de cada patrón. Sin embargo, [Larman98] enfatiza que los patrones "describen principios fundamentales de asignación de responsabilidades a los objetos" en los diagramas de interacción, y no explica cómo debe traducirse esa asignación de responsabilidades llegado el momento de la codificación.

Diagrama de clases

Con los diagramas de colaboración terminados se esta en posibilidad de identificar las especificaciones de las clases de software que serán parte de la solución del problema. La creación del diagrama de clases toma en consideración los conceptos del mundo real descubiertos en el dominio del problema, los cuales han sido expresados en el modelo conceptual. Varios de estos conceptos se traducen a una clase de software, al ser identificados como tales por medio de los diagramas de colaboración, estos diagramas también proporcionan información sobre los métodos que formaran parte de la clase, estos métodos son identificados por los mensajes intercambiados por las clases cuando se modela una operación del sistema por medio de los diagramas de interacción.

Para los que hemos sido solamente usuarios de las bibliotecas de clases y no hemos escrito nunca una clase, resulta problemático realizar el signado de ellas así como establecer las relaciones que se deben dar entre ellas. Para poder superar esta dificultad nos fue necesario codificar la definición e implementación de algunas clases que debían formar parte del diagrama, aunque estabamos conscientes de que esta codificación tal vez no sirviera para nada el futuro, no consideramos que fue tiempo perdido porque ello nos dio el conocimiento y experiencia de lo que es realmente una clase y de las partes que la conforman.

La conclusión del diagrama de clases marca el final de la fase de diseño, para dar inicio a la última fase del proceso de desarrollo que es la Construcción.

Construcción

El llamarle a esta fase Construcción puede causar un poco de confusión con el término Construir utilizado para referirse al segundo paso de nivel macro del proceso de desarrollo. Sin embargo, la fase de Construcción se refiere específicamente a la etapa del proyecto donde se genera el código del sistema, es decir, la implementación o codificación en algún lenguaje orientado a objetos como puede ser Java o C++.

Del diagrama de clases creado se tiene mucha información la cual puede traducirse directamente a código. Por ejemplo, para la definición de la clase se toman los elementos que existen en cada uno de los tres compartimentos del cuadro que representa la clase, del primero se toma el nombre de la clase, del segundo o intermedio, los atributos y del último los nombres de los métodos que tiene la clase.

Para los que no tienen experiencia en el modelo de objetos, el proceso de desarrollo los conduce de una manera muy natural y lógica a la definición de la clase y sus métodos con algún lenguaje de programación. Además de esto, en esta fase también se muestra como las relaciones entre clases se traducen a atributos de referencia en la codificación.

Definida la clase, sólo resta definir los métodos de la misma. Para esto se toman los mensajes de los diagramas de colaboración, los cuales se envían como respuesta cuando se invoca un método. El orden en el que se dan estos mensajes se traduce a una expresión en un lenguaje de programación, esta expresión es lo que se conoce como una declaración de software.

Para hacer la construcción del sistema, ha sido necesario simular la responsabilidad que tiene la Entidad de Aplicación de solicitar a DIMSE alguno de los servicios que proporciona; para esto se ha convenido que tanto la primitiva de petición como de respuesta para un servicio de intercambio de mensaje DICOM, tengan un solo argumento, este argumento es un buffer de caracteres que lleva al principio el nombre del servicio solicitado o respondido, seguido por los datos que marca el Estándar como obligatorios para el servicio en cuestión. Para delimitar cada uno de los componentes de este buffer se ha establecido un carácter como delimitador de los mismos.

Del diagrama de clases se han definido en código de lenguaje de C++ Builder cada una de las clases que la conforman, así como los métodos que se requieren para atender a una primitiva de petición del servicio C-ECHO. Al tiempo de estar escribiendo esta parte , se esta trabajando en la codificación del mensaje por parte de DIMSE; esto se esta haciendo como una prueba más del diseño de solución planteado por el diagrama de clases obtenido. Hasta lo que se lleva codificado, el diagrama de clases ha proporcionado toda la información necesaria para codificar la petición el servicio C-ECHO, por tal razón se espera y confía que los demás servicios se puedan obtener de la misma manera, puesto que el diseño los ha contemplado a cada uno de ellos.

6. Conclusiones y Perspectivas

DICOM es un estándar complejo. Para hacer aun más difíciles las cosas para aquellos que se encuentran con él por primera vez, esta escrito en el lenguaje árido de los estándares y con un mínimo de información explicativa. No obstante, se logró comprender el tema correspondiente al intercambio de mensajes DICOM, auxiliándose y apoyándose en la metodología propuesta por el proceso de desarrollo de proyectos de software.

Seguir un proceso de desarrollo obliga a organizar de manera ordenada las actividades que se deben realizar durante el mismo. Define el punto de partida y la

secuencia de aplicación de las fases para un ciclo de desarrollo, y es a través de esta dirección que se van obteniendo los productos necesarios para crear un modelo de solución del dominio del problema en cada una de las fases.

Al inicio del proyecto, se contaba con muy poca información y experiencia tanto en el proceso de desarrollo como en el protocolo DIMSE, a medida que se iteraron las diversas fases del proyecto se fueron supliendo estas carencias, afirmando y dando seguridad en los resultados obtenidos.

El proceso de desarrollo establece herramientas y actividades a utilizar y realizar respectivamente, las cuales no deben emplearse de manera rígida y absoluta porque no existe el proceso perfecto que garantice el mejor diseño y éxito de un proyecto de software. Del proceso de desarrollo deben utilizarse aquellas herramientas y realizarse aquellas actividades que le proporcionen al desarrollador los conocimientos y claridad en el dominio del problema para crear el modelo de solución. En mi experiencia, el desarrollador no debe realizar más actividades, marcadas por el proceso, si tiene resueltas todas las interrogantes del problema y es capaz de explicar cabalmente el comportamiento del sistema en desarrollo.

DIMSE es sólo la parte encargada de suministrar los servicios para que una implementación de DICOM intercambie un mensaje. Así es de que, como proyecto, no es en si un todo, sino que forma parte de un proyecto mayor que se pretende siga el modelo de capas. En este sentido, DIMSE, debe comunicarse con su capa inferior que es DUL (DICOM Upper Layer), que es, hasta el momento la capa más avanzada en su desarrollo. Hacia arriba, ya no existe propiamente una capa, porque DIMSE forma parte de la Entidad de Aplicación DICOM, sin embargo, dentro de ella se comunica hacia arriba con las Clases de Servicio y los Objetos de Información, que es otro aspecto del proyecto que actualmente también se encuentra bajo desarrollo.

Tanto DUL como DIMSE han sido terminados hasta su etapa de diseño, lo que sigue es terminar la codificación de ambos protocolos, a fin de acoplarlos y probar sus interfases e interoperabilidad entre ellos, realizando, de ser necesario, las adecuaciones pertinentes para su correcto funcionamiento. Se espera que los problemas que se puedan presentar sean mínimos, dado que en ambos proyectos se ha utilizado el modelo de objetos para su desarrollo y aplicando procesos de desarrollo, que si bien no son idénticos, son muy semejantes.

7. Bibliografía

Ambler97 Ambler, Scott W.; The Unified Modeling language v1.1 and Beyond: The Techniques of Object-Oriented Modeling, http://www.ambysoft.com/umlAndBeyond.pdf, 1997.

Berard95 Berard, Edward V.; Be Careful With "Use Cases", The Object Agency, Inc., 1995.

Booch96 Booch, Grady; Object Solutions: Managing the Object-Oriented Project, Addison-Wesley, 1996.

Booch94 Booch, Grady; Análisis y diseño orientado a objetos con aplicaciones, Addison-Wesley / Diaz de Santos, 2ª ed., 1996.

Jacobson99 Jacobson, I.; Booch G.; Rumbaugh J.; The Unified Software Development Process, Addison-Wesley, 1999.

Jacobson92 Jacobson, I.; Object-Oriented Software Engineering: A Use Case Drive Approach, Addison-Wesley, 1992.

Larman98 Larman, Craig; Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design, Prentice Hall PTR, 1998.

Leotta93 Leotta, D. F., Kim, Y.; "Requirements for Picture Archiving and Communications. Electronic Handling of Image Data Needs Clinical Acceptance for Success", IEEE Engng in Med and Biol, pp. 62-69, March 1993.

NEMA37 NEMA Standards Publication PS 3.7-1993. Digital Imaging and Communications in Medicine (DICOM) Part 7: Message Exchange, National Electrical Manufacturers Association, 1994.

NEMA38 NEMA Standards Publication PS 3.8-1993.

Revet97 Revet, B.; "DICOM Cook Book for Implementations in Modalilities", Philips Medical Systems Nederland B. V., Version 1.1, January 1997.

Steven Steven C. Horiil, Fred W. Prior, W. Dean Bidgood, Jr., Charles Parisot, Geert Claeys; "DICOM: An Introduction to the Standard"

Land Land, CD, Lashin V, Oriol A, Villanueva JJ; Object-oriented Design of the DICOM Standard and its Application to Cardiovascular Imaging, http://www.cvc.uab.es/~craig/dicom/oo-dicom.htm