UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Unidad Académica de los Ciclos Profesional y de Posgrado

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

Desarrollo del Protocolo de Capa Superior del Estándar DICOM Utilizando Metodología Orientada a Objetos

T E S I S

Que para obtener el grado de

MAESTRO EN CIENCIAS DE LA COMPUTACION

p r e s e n t a

ALFONSO MARTINEZ MARTINEZ

Directora: Dra. Verónica Medina Bañuelos

México, D. F. **Mayo d** **1999**

Como hay una palabra para designar la sabiduría,
algunos creen saber lo que es la sabiduría.
Pero nadie llega a ser un astrónomo
por haber comprendido el significado
de la palabra "astronomía".

Anthony de Mello

A Imelda,
Andrés y
...

AGRADECIMIENTOS

- A mi directora de tesis, Dra. Verónica Medina, por su gran apoyo y paciencia en el desarrollo de este trabajo y del proyecto PACS.
- A la Dra. María Garza, por el apoyo recibido como alumno de la maestría y por la revisión de este trabajo.
- A la M. en C. Guadalupe Ibargüengoitia por su apoyo en la revisión de este trabajo.
- A la Dra. Hanna Oktaba, por generar en mi la inquietud en el estudio de sistemas orientados a objetos y por su apoyo en la revisión de este trabajo
- Al Dr. Manuel Romero, por su disponibilidad, críticas y sugerencias que contribuyeron a mejorar este trabajo.
- A todos los miembros del área de investigación PDSIB-LICPOS, en la UAM-Iztapalapa, por su gran apoyo y compañerismo.

Contenido

1. INTRODUCCIÓN

1.1. Necesidades en el manejo de imágenes médicas a gran escala

Las necesidades de almacenamiento y manipulación de imágenes médicas surge a partir de los años 70's como consecuencia del nacimiento de la tomografía computarizada (*CT-Computed Tomography*) como método de diagnóstico basado en imágenes digitales. Desde entonces, se han desarrollado diferentes técnicas en la obtención de imágenes como la medicina nuclear (*NM-Nuclear Medicine*), la resonancia magnética (*MR-Magnetic Resonance*), la radiografía computarizada (*CR-Computed Radiography*) y la angiografía por substracción digital (*DSA-Digital Subtraction Angiography*), entre otras. Estas técnicas han contribuido a la generación de diferentes tipos de imagénes médicas digitales para diagnóstico, junto con el consecuente incremento en la producción de las mismas [Leotta93]. Esto ha complicado el manejo de las imágenes principalmente en la impresión y almacenamiento posterior, produciendo una gran demanda de medios de almacenamiento más apropiados (no impresiones en papel o en placas radiográficas) y, a su vez, métodos de transferencia entre dispositivos manufacturados por diferentes compañías [Clunie95].

Para el caso de las imágenes médicas, además de los atributos de la imagen y de la imagen misma, normal o comprimida, se agregan datos demográficos y de identificación del paciente, información acerca de las condiciones de adquisición y, en algunos casos, información del examen, serie a la que pertenece la imagen y orden que guarda en un estudio. Por lo tanto, es necesario contar con sistemas de información que ofrezcan una alternativa en el manejo de imágenes médicas a gran escala, facilitando todas las actividades relacionadas con las mismas en beneficio de los pacientes de un hospital.

1.2. Los Sistemas PACS (*Picture Archiving and Communication Systems*)

Los sistemas PACS ([Dale88], [Allen92], [Leotta93] y [Mun93]) ofrecen una alternativa en el manejo de imágenes digitales en forma eficiente y a gran escala,

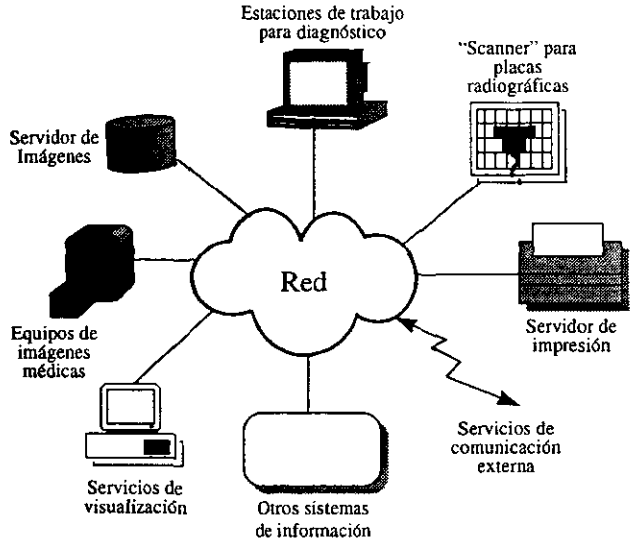


Figura 1.1: Sistemas PACS

a través de dispositivos conectados en red (ver fig. 1.1). El conjunto de estos dispositivos ofrecen una serie de servicios que dan soporte a la operatividad de un área (radiología en el caso de aplicaciones médicas).

Entre las características que los sistemas PACS deben ofrecer para obtener una buena aceptación en el medio clínico, se deben considerar: la facilidad, rapidez, seguridad en el acceso de imágenes y la calidad en su presentación. Además, se pueden aprovechar las facilidades de la tecnología computacional para ofrecer funciones adicionales como mostrar varias imágenes en una misma pantalla, procesar imágenes para corregirlas o mejorarlas, grabar voz correspondiente al diagnóstico y realizar diagnóstico asistido por computadora, entre otras.

Cabe señalar que el uso de los sistemas PACS no es exclusivo del área médica; existen otras áreas que requieren de la manipulación de grandes cantidades de imágenes, como: la Geología, la Geografía y el estudio de fenómenos atmosféricos, en donde básicamente se utilizan imágenes de percepción remota.

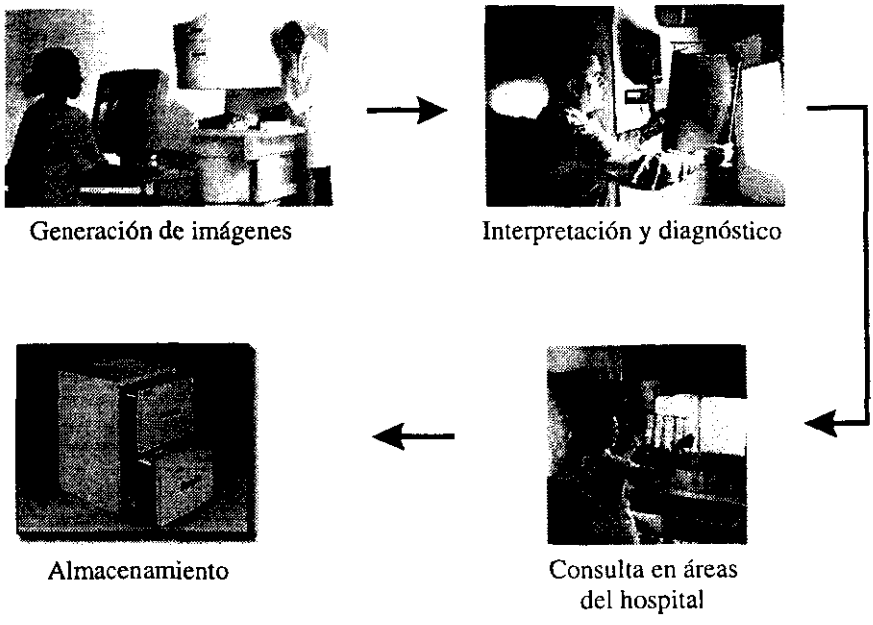


Figura 1.2: Manejo y uso tradicional de imágenes médicas

1.2.1. Beneficios

Actualmente, la mayoría de los hospitales de la Ciudad de México, de regular tamaño, trabajan manualmente las imágenes médicas de diagnóstico (ver figura 1.2). En algunas modalidades como rayos-X (*XR-X Ray*), CT y MR, las imágenes son impresas en placa radiográfica y en otros casos, como el ultrasonido, son grabadas en cinta de video. Esta información debe pasar por varias etapas para su interpretación, diagnóstico y consulta posterior. Durante su análisis, comúnmente el médico especialista utiliza un medio de grabación de voz para registrar su diagnóstico, que después es transcrito por otra persona en un procesador de texto o máquina de escribir. Posteriormente, los resultados de ese diagnóstico son utilizados dentro del hospital en varios servicios para ofrecer un tratamiento apropiado a la patología del paciente. Como se puede inferir, existen posibilidades de error en el diagnóstico debido al manejo de la información; por ejemplo, si se maltrata la imagen, si la transcripción no se realiza apropiadamente o, aún más grave, si se extravía la información.

Por lo regular, cuando el paciente abandona el hospital, las imágenes impresas se almacenan en el área de archivo clínico, en donde es difícil mantener un orden

apropiado, debido a la gran cantidad de placas a archivar. Como consecuencia, cuando un médico requiere consultar alguna imagen del archivo para realizar comparaciones con patologías similares, la probabilidad de conseguirlas es muy baja.

Las ventajas que un sistema PACS ofrece sobre un manejo manual de imágenes son [Leotta93]:

- Se elimina la necesidad de procesamiento de placas impresas y los costos asociados (en material, personal y tiempo).
- Se elimina el espacio físico ocupado por imágenes impresas, debido al almacenamiento electrónico usado en PACS.
- Se ahorra tiempo en los procesos de recuperación de imágenes.
- Se tiene una probabilidad de pérdida de prácticamente cero al establecer un esquema de respaldo en la información, en donde intervienen discos magnéticos, discos ópticos y unidades de cinta, como dispositivos de almacenamiento.
- Se obtiene una organización eficiente en estudios que requieren varias imágenes (por ejemplo, almacenándolas en un solo archivo).
- Se facilita la consulta de imágenes, de tal forma que, una imagen puede ser observada por varios usuarios en lugares distintos al mismo tiempo.
- Se facilita el trabajo de diagnóstico, si los datos de una imagen pueden ser mejorados realizando algún tipo de procesamiento.
- Se obtiene la facilidad de transmitir una imagen a un lugar remoto del hospital vía una red de telecomunicación.
- Se pueden mantener en el sistema imágenes con mucho tiempo de antigüedad [Allen92], con un esquema de almacenamiento apropiado para el sistema.
- Se tiene la posibilidad de hacer comparaciones de imágenes de diferentes pacientes, con el mismo padecimiento, aún cuando se trate de imágenes con algunos años de antigüedad.
- Se puede incorporar la información de diagnóstico a las imágenes.
- Se puede integrar el sistema PACS a otros sistemas de información del hospital, tales como: el sistema de información hospitalario (*HIS-Hospital Information System*) o el sistema de información de radiología (*RIS-Radiology Information System*).

- Se tiene la posibilidad de consultar bancos de imágenes con patologías similares con la finalidad de apoyar la capacitación y la enseñanza de radiólogos en formación.

Podría pensarse que la instalación de un sistema PACS es inalcanzable, debido a la gran cantidad de recursos que requiere. Sin embargo, la relación costo/beneficio del sistema es rentable a largo plazo por el ahorro que podría obtenerse en otros rubros. Por ejemplo, el proceso de impresión y el personal asignado a éste, entre otros.

1.2.2. Requerimientos

Los sistemas PACS están integrados por un conjunto de dispositivos, cuyas responsabilidades son las de ofrecer todos los elementos operacionales demandados por el área de radiología y áreas dependientes. Estas demandas incluyen:

Adquisición de imágenes. Varias modalidades son de naturaleza digital (CT. CR, MR, NM, DSA, entre otras). Sin embargo existen modalidades cuya información analógica requiere de digitalizadores especiales (Ultrasonido y XR, entre otros).

Almacenamiento de información. Una vez teniendo la información digital, se requiere almacenarla utilizando manejadores de bases de datos adaptados al manejo de imágenes. Debe contemplarse además, el almacenamiento de datos asociados (información del paciente o de los estudios realizados). Este trabajo se puede realizar en ambientes centralizados o distribuidos.

Transferencia local de imágenes. Desde el lugar donde se producen las imágenes, se requiere consultarlas en diferentes áreas en el hospital con la finalidad de realizar interpretación, dignóstico o alguna revisión. Para ello, se requiere una red de comunicación que por un lado, integre a los diferentes dispositivos involucrados y por el otro, permita el envío de imágenes a las diferentes áreas del hospital.

Consulta de imágenes. Las imágenes necesitan visualizarse con propósitos de revisión, interpretación y diagnóstico. En cada situación, se deben utilizar dispositivos apropiados. Los casos más delicados son la interpretación y el diagnóstico, ya que requieren de estaciones de trabajo con posibilidades gráficas importantes que conserven los detalles de una impresión normal en placa. También, se debe contemplar la posibilidad de resaltar algunos

aspectos de las imágenes con el fin de facilitar el trabajo del radiólogo o especialista que lo requiera. Esto se logra utilizando procesamientos específicos de las imágenes. Para el caso de la revisión, debe existir la posibilidad de mostrar también los resultados del diagnóstico e interpretación para cada imagen y, si es el caso, sincronizar la voz con el despliegue secuencial de las imágenes.

Registro de resultados. El resultado de la interpretación y diagnóstico debe integrarse a las imágenes y a su vez al sistema PACS, ya sea a través de voz o texto.

Interfase con otros sistemas. Los sistemas más antiguos en los hospitales son el HIS y en ocasiones el sistema RIS. Estos sistemas tienen definido cierto tipo de información que también es utilizado por los PACS. Para evitar redundancia en la información, se debe contemplar una interfase entre ellos. Con esta interfase, se puede optimizar la utilización de recursos, mejorar la calidad de servicios al paciente, minimizar la dependencia en la impresión de imágenes y dar facilidades a la investigación y soporte a la educación médica.

Transferencia remota de imágenes. Otro aspecto a contemplar en el desarrollo de un sistema PACS, es la posibilidad de intercambiar información entre hospitales. Por ejemplo, cuando un paciente es trasladado de un hospital a otro se podrían consultar sus estudios realizados, en forma remota, para no exponerlo a más radiación. Esto requiere de la utilización de un mecanismo de conexión entre redes.

Disponibilidad y escalabilidad. El sistema debe estar disponible para proporcionar servicio las 24 horas del día, durante los 365 días del año. Para cumplir con este requerimiento, se debe redundar en canales de comunicación, utilizar de preferencia bases de datos distribuidas y algún esquema de protección de información en cinta. El sistema debe ser escalable, es decir, lo suficientemente flexible para aceptar modificaciones y adiciones en su arquitectura, sin afectar el servicio. Las operaciones de mantenimiento deben ser minimizadas y automatizadas y se debe tener alguna capacidad de monitoreo del sistema.

1.2.3. Componentes

Como puede inferirse de los requerimientos, los sistemas PACS utilizan varios componentes (hardware y software) con funciones específicas. Estos componentes

son: digitalizadores laser para placas de XR, digitalizadores de video, estaciones de trabajo con diferentes características, estaciones de consulta, medios de almacenamiento óptico y magnético, servicios de impresión, infraestructura para servicios de red, servidores de imágenes, servidores de bases de datos, dispositivos que generan imágenes médicas digitales y servicios de comunicación a sistemas remotos externos, entre otros.

Como se observa en la figura 1.1, estos componentes se integran en un esquema Cliente/Servidor para ofrecer los diferentes servicios demandados por el área de radiología.

1.3. Estandarización en el manejo de imágenes médicas

En el campo de la representación digital de imágenes existe toda una gama de formatos utilizados para su almacenamiento y manipulación. Varios de ellos surgieron de programas para computadoras personales (BMP-Windows bitmap o PICT-Macintosh) o como consecuencia de necesidades de intercambio de información en algunos ambientes de trabajo (JPEG-Joint Photographics Expert Group, MPEG-Motion Pictures Expert Group, GIF-Graphics Interchange Format/CompuServe). Debido a la independencia de los dispositivos utilizados para su creación, que algunos formatos guardan, en algunos casos se ha intentado definir estándares a través de ellos para ambientes de trabajo específicos.

Algunos estándares definen la información de una imagen como arreglos de pixeles (bitmap) que se pueden mapear directamente a la pantalla de un monitor o a una impresora. Otros definen a cada objeto, que forma parte de la escena de una imagen, vectorialmente en un sistema de coordenadas (por ejemplo: CDR-Corel Draw, PLT-Hewlett Packard). En un formato se integran varios elementos de información, como los atributos de la imagen (tamaño, niveles de gris, mapa de colores, etc.) e información relacionada al contexto de uso. Estos elementos se almacenan en una sección de encabezado además de la información propia de la imagen, que en ocasiones es comprimida.

Las características que cada formato posee, lo hace apropiado al contexto para el que fue creado. Sin embargo, los formatos no mantienen restricciones en cuanto al tipo de imágenes que manipulan y se debe considerar la posibilidad de degradación en la calidad de la imagen de acuerdo a la forma específica de manejo establecida. Por lo tanto, es importante conocer la estructura interna del formato para entender la organización de cada elemento de información. Para ello, se pueden considerar 3 tipos diferentes [Clunie95]:

Formato fijo. En este formato, la representación de la información es idéntica en cada archivo. Define un grupo de campos de longitud fija en la misma

posición para almacenar la información del encabezado y los datos de la imagen. Los datos correspondientes a la imagen son los únicos que pueden cambiar su longitud.

Formato en bloque. En este formato, el encabezado está formado por un grupo limitado de apuntadores a bloques de información. Los bloques por lo general se encuentran en el mismo lugar y son de longitud constante.

Formato con etiquetas. En este formato, cada elemento de información se define utilizando etiquetas. Cada etiqueta es seguida de la longitud del elemento y de la información. De esta forma, cada elemento de información es considerado como autocontenido y autodescriptivo.

En un principio, para el caso de las imágenes médicas digitales, cada fabricante de equipos generadores de imágenes médicas realizó su propia especificación para el manejo de información. De esta forma, se estableció la posibilidad de intercambio sólo entre dispositivos de la misma marca y en algunos casos del mismo modelo. Dentro de las especificaciones se utilizaron básicamente formatos fijos o de bloque, algún tipo de codificación especial, compresión y formato especial para almacenamiento en cinta para intercambio de información con propósitos clínicos o de investigación. Todo esto llevó a la definición de procedimientos particulares para la manipulación de las imágenes en cuanto a adquisición, visualización, almacenamiento e intercambio de información. A su vez, hasta hace algunos años, el desarrollo tecnológico en cuanto medios de almacenamiento, procesamiento y servicios de comunicación con imágenes digitales, no tenía aún el desarrollo apropiado para soportar eficientemente las necesidades de manipulación. Actualmente, las tecnologías emergentes en hardware superan la problematica planteada anteriormente, favoreciendo a la comunidad médica con dichos servicios. Sin embargo, surgen otro tipo de problemas al tratar de integrar los distintos formatos de imágenes digitales particulares a cada fabricante.

Como ejemplo [Clunie95], se tiene el equipo de CT General Electric CT9800 que utiliza un formato de bloque para sus imágenes digitales. Los bloques están formados por series de 256 palabras de 16 bits. El bloque 0 contiene el encabezado general con apuntadores a otros encabezados, como los de examen e imagen, además de información relacionada al número de bloques utilizado por los encabezados y los datos de la imagen. Los datos pueden almacenarse de dos formas, lo cual depende de si se selecciona compresión o no. Si se elige compresión, se utiliza DPCM (*Differential Pulse Code Modulation*). Si las imágenes se almacenan sin compresión es posible recuperar la información utilizando medios externos al sistema. Sin embargo, al guardar la información comprimida es prácticamente

imposible hacer esto. El equipo también incluye una unidad de cinta magnética de 9 pistas que utiliza su propio formato de almacenamiento. Las características particulares de este equipo se complementan con estaciones de trabajo que incluyen software muy específico para la visualización y manejo de imágenes.

En las unidades de radiología es muy común encontrar equipos de diferentes marcas para las diferentes técnicas en la generación de imágenes. El tratar de integrar todos ellos en un sistema para su almacenamiento y transferencia es prácticamente imposible. Como consecuencia, surgió la necesidad de estandarizar el proceso de manipulación de imágenes médicas digitales. Este trabajo se inició en 1983 con la integración de un comité formado por el Colegio Americano de Radiología (*ACR-American College of Radiology*), en representación de la comunidad de radiólogos y la Asociación Nacional de Manufactura Eléctrica (*NEMA-National Electrical Manufacturers Association*), que agrupa a la industria en el área de radiología, de acuerdo a los procedimientos establecidos por NEMA. Los objetivos iniciales fueron:

- Promover la comunicación entre imágenes digitales independientemente del fabricante que las produjo, considerando así el problema de compatibilidad.
- Ofrecer mayor flexibilidad a los sistemas de almacenamiento y transferencia de imágenes.
- Facilitar la creación y consulta de imágenes a través de diferentes dispositivos, en diversos lugares locales o remotos, con propósitos de visualización.

1.3.1. Evolución

Los primeros resultados en los trabajos de estandarización fueron publicados en 1985 ([Horiil95] y [Hintel94]) bajo el nombre de ACR-NEMA Versión 1.0, teniendo como base las ideas obtenidas de formatos ya existentes. Por ejemplo, la definición de elementos de datos de longitud variable e identificados con etiquetas [Clunie95] fue adoptada de un estándar para grabar imágenes en cinta magnética, desarrollado por la Asociación Americana de Físicos en Medicina (*AAPM-American Association of Physicists in Medicine*). Sin embargo, como todas las primeras versiones se detectaron varios errores. De esta forma, el comité encargado (*ACR/NEMA*) autorizó, a los grupos de trabajo involucrados, la realización de dos revisiones en Octubre de 1986 y en Enero de 1988, que produjeron la versión ACR-NEMA Versión 2.0 en 1988.

En esta nueva versión, se conservaron prácticamente las mismas especificaciones de interfase con hardware definidas en la versión 1.0, pero se agregaron

nuevos elementos de datos y se corrigieron varios errores e inconsistencias relacionadas a tipos de datos. En esta versión, se especificó la comunicación punto a punto entre dispositivos, un grupo de comandos por software y varios formatos de datos correspondientes a nuevos elementos.

En el tiempo que se dió a conocer la segunda versión, surgió la demanda de interfase entre dispositivos involucrados en la generación de imágenes y redes de cómputo. Sin embargo, el estándar ACR-NEMA Versión 2.0 no ofrecía ningún soporte de comunicación en red. La respuesta a estas demandas implicaba grandes cambios a lo ya establecido, considerando como restricción principal la de mantener la compatibilidad con las versiones anteriores. Este hecho fue un gran reto para los grupos de trabajo. De esta forma, a partir de 1988, se comenzó a trabajar en una tercera versión, en donde el proceso de diseño sufrió un cambio radical porque se incluyeron modelos para:

- Simulación del *mundo real* a través de diagramas *entidad-relación* para modelar los aspectos más importantes de las áreas de radiología.
- Interoperabilidad de *sistemas heterogéneos* en ambiente de red a través de protocolos de comunicación especificados.
- Establecimiento de asociaciones a través de envío de mensajes entre dispositivos compatibles, bajo el modelo *Cliente/Servidor*.

Después de tres años de esfuerzo, se dió a conocer la versión ACR/NEMA DICOM (*Digital Imaging and Communications in Medicine*) llamada también DICOM 3.0 [NEMA93]. En esta versión participaron varias instituciones de la comunidad internacional como la Industria Japonesa de Aparatos de Radiología (JIRA-Japanese Industry Radiology Apparatus) y el Comité Europeo de Normalización (CEN-Comité Européen de Normalisation). Esta versión es considerada como un estándar completo y compatible porque supera las deficiencias de sus predecesores y acepta las especificaciones de las versiones anteriores.

1.3.2. El estándar DICOM (*Digital Imaging and Communications in Medicine*)

Las principales diferencias que tiene DICOM con respecto a las versiones anteriores son:

- Intercambio de información en redes de comunicación y en medios de almacenamiento al especificar comandos definidos por una sintaxis y una

semántica asociados con datos. Para complementar estas actividades, DICOM especifica varios protocolos de comunicación en red para ofrecer servicios a varios niveles. Las versiones anteriores sólo ofrecían comunicación punto a punto.

- Especificación de diferentes niveles de compatibilidad al establecer explícitamente un determinado nivel de compatibilidad, indicando sólo opciones específicas ofrecidas por DICOM (por ejemplo, establecer sólo el manejo de imágenes de CT). En las versiones anteriores únicamente se especifica un nivel mínimo.

- Información explícita de objetos a través de las especificaciones de estructuras de datos que facilitan la manipulación de objetos como entidades autocontenidas. Estos objetos no son únicamente imágenes y gráficas, sino también estudios, reportes, etc.

- Identidad única de objetos como consecuencia de aplicar el modelo orientado a objetos a las especificaciones establecidas en el manejo de la información. De esta forma, se pueden obtener instancias (u objetos) con operaciones permitidas y definidas a través de clases.

- Flexibilidad al ofrecer la posibilidad de definir nuevos servicios no estándar.

- Interacción eficiente entre servicios ofrecidos y aplicaciones a través de una configuración definida por el estándar, que establece una comunicación eficiente entre el usuario de servicios y el proveedor de los mismos.

- Representación de aspectos del mundo real modelados a través de diagramas *entidad-relación*. Esto permite la especificación de objetos compuestos que describen un contexto completo y objetos normalizados como entidades del mundo real.

- Documentación estandarizada al apegarse a las normas de la Organización Internacional para Estandarización (*ISO-International Organization for Standarization*) en la estructura de una documentación multi-partes. De esta forma, facilita su evolución simplificando la adición de nuevas partes.

El principal beneficio obtenido con estos servicios en un hospital es poder comunicar el sistema de información para el manejo y comunicación de imágenes (*PACS*), el sistemas de información del servicio de radiología (*RIS*) y el sistema de información administrativo (*HIS*).

1.4. Comunicación en red para aplicaciones DICOM

1.4.1. El modelo de capas para protocolos de comunicación

Un protocolo de comunicación representa los formatos de mensajes y las reglas que dos o más máquinas deben seguir para poder intercambiar estos mensajes. Las aplicaciones en red utilizan protocolos para comunicar a dos computadoras o más y tienen asignado un trabajo muy específico. tanto los sistemas complejos de comunicación, como las aplicaciones desarrolladas con el estándar DICOM, no utilizan un solo protocolo para realizar sus trabajos de transmisión, sino que requieren varios de ellos trabajando conjuntamente. Estos protocolos se pueden representar en un modelo de capas, en donde cada capa representa a uno o más protocolos. Concretamente se tiene el *modelo de referencia ISO/OSI (International Organization for Standardization/Open Systems Interconection)* y al *conjunto de protocolos de TCP/IP (Transmission Control Protocol/Internet Protocol)* como representativos de este modelo.

Cada capa mantiene su funcionalidad de acuerdo a sus especificaciones. Cada una de las capas adiciona algún valor a la funcionalidad de la de abajo, lo cual significa que utiliza los servicios de la capa de abajo para construir un servicio más enriquecido. Los beneficios que ofrece el modelo de capas son la definición independiente y simultánea de cada capa y la posibilidad de reuso para las aplicaciones, sin tener que reinventar funciones y mecanismos.

Las tareas asignadas a un conjunto de protocolos en una aplicación son [Comer95]: tolerar a fallas de hardware o del sistema operativo de un sistema remoto, manejar el tráfico de paquetes de información ante una congestión en red, manejar el retraso o la pérdida de paquetes de información, tomar decisiones ante la corrupción de datos y corregir errores de duplicidad o de secuencia en datos enviados por red. Sería difícil que un sólo protocolo resolviera toda esta problemática, por lo que en el modelo de capas, cada capa (representada por uno o más protocolos) toma como responsabilidad el manejo de una parte del problema.

Cada capa toma decisiones acerca de la forma correcta de manejo del mensaje y selecciona una acción apropiada, de acuerdo al tipo de mensaje y su destino. El objetivo del modelo de capas en protocolos de comunicación es hacer una abstracción de los elementos que intervienen en la comunicación, de acuerdo a su funcionalidad y sus formatos de datos.

1.4.1.1. El modelo de referencia ISO/OSI

En el caso del modelo de referencia ISO/OSI [Larmouth94] se definen 7 capas con protocolos (reglas) y servicios estándar (notación), asignados a cada una de ellas. El principal objetivo expresado en la definición del modelo de referencia ISO/OSI es garantizar una comunicación entre computadoras, independiente tanto de aspectos que dependan de la aplicación, como de aspectos tecnológicos (señalización y ruteo). Un resumen breve de las 7 capas de este modelo se presenta a continuación.

Capa Física. Establece la interconexión física (paralela o serial) entre computadoras y conmutadores de paquetes de red, así como los procedimientos utilizados para transferir paquetes de información entre una máquina y otra, incluyendo especificaciones eléctricas. La información manipulada a este nivel son cadenas de bits.

Capa para la Liga de Datos. Define el formato de la trama de datos (*frames*) y especifica cómo dos máquinas reconocen las fronteras de estas unidades. También incluye detección de errores y mecanismos de retransmisión hasta confirmar una transferencia exitosa.

Capa de Red. Define la unidad básica de transferencia a través de la red e incluye los conceptos de dirección destino y ruteo, en contextos de redes a gran distancia (*WAN-Wide Area Network*). Esta capa ensambla un paquete en la forma requerida por la red y utiliza la capa inferior para transferirlo (posiblemente fragmentado) al conmutador de paquetes. También debe responder a problemas de congestión en la red.

Capa de Transporte. Ofrece confiabilidad y eficiencia de extremo a extremo entre dos máquinas en la red. Aunque las capas inferiores realizan pruebas de confiabilidad en cada transferencia, la capa de transporte también realiza pruebas de confiabilidad para asegurar que no existen fallas en la red.

Capa de Sesión. En esta capa, se establece control y separación de diálogos para flujo de datos en una o ambas direcciones al mismo tiempo. También se encarga de la sincronización del flujo de información, estableciendo puntos de revisión para no retransmitir mensajes completos en caso de fallas.

Capa de Presentación. Esta capa se encarga del manejo de la información transmitida definiendo una semántica, estructura y representación de bits para los mensajes enviados. Esto se traduce en la definición de *sintaxis abstracta* y *sintaxis de transferencia* para dichos mensajes.

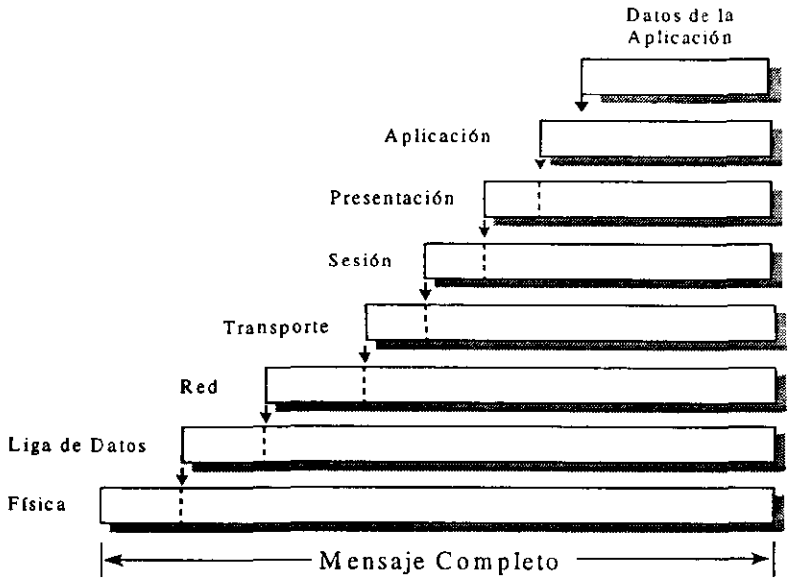


Figura 1.3: Estructura de mensajes en el modelo de referencia ISO/OSI.

Capa de Aplicación. En esta capa, se contemplan varios protocolos de aplicación para la utilización de la red a través de elementos de servicio de aplicación (*ASE-Application Service Element*). También, se definen *Entidades de Aplicacion (AE-Application Entity)* que hacen uso de los servicios que ofrece la capa inmediata inferior. Actualmente, se ha establecido para la capa de aplicación ISO/OSI la utilización de los servicios establecidos por ACSE (*Association Control Service Element*) para controlar las asociaciones entre aplicaciones.

En la figura 1.3, se presenta la forma en la que se organizan los mensajes para el modelo de referencia ISO/OSI. Los datos a enviar por alguna aplicación son pasados a la capa de aplicación quien le agrega información particular para el control del mensaje en ese nivel. Lo mismo sucede en la capa de presentación y en las demás capas superiores hasta formar un mensaje completo que viajará por la red. Cabe señalar que la complejidad del mensaje en cada capa no es tan simple como lo expresa la figura.

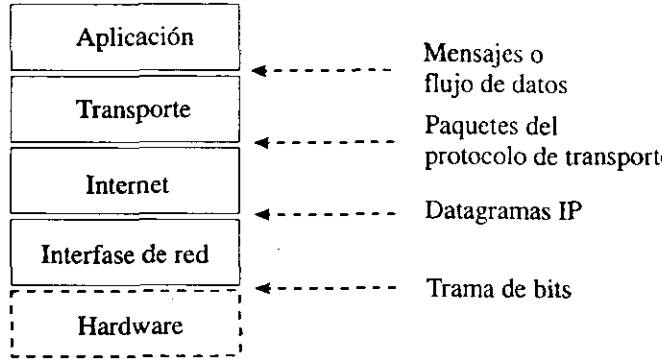


Figura 1.4: Capas conceptuales de TCP/IP y la forma de manejo de mensajes entre capas.

1.4.1.2. Los protocolos TCP/IP

Estos protocolos están organizados en 4 capas conceptuales de software, junto con una quinta capa que representa el hardware. Se podría considerar que con un poco de trabajo, se puede reducir el modelo de referencia ISO/OSI para describir el esquema de capas de TCP/IP [Comer95]. En la figura 1.4 se observan estas 4 capas y la forma de manejo de datos se describen enseguida.

Capa de Aplicación. A este nivel, se invoca a programas de aplicación que accesan a los servicios de TCP/IP. Las aplicaciones interactúan con uno de los protocolos de la capa de transporte para enviar y recibir datos. Cada aplicación selecciona la forma de transporte requerido. Si se requiere confiabilidad en los datos enviados, se puede solicitar una transferencia orientada a conexión utilizando el conjunto de protocolos TCP, en otro caso puede hacer uso de los protocolos UDP (*User Datagram Protocol*).

Capa de Transporte. La capa de transporte regula el flujo de información con alta confiabilidad, asegurando que los datos lleguen a su destino sin error y en la secuencia apropiada. Para transmitir la información, el transporte divide el flujo de datos en piezas pequeñas (llamadas *paquetes*) que son pasadas a la siguiente capa con su dirección destino para su transmisión.

Capa de Internet (Internet Protocol, IP). Maneja la comunicación entre dos máquinas conectadas en una red. Encapsula los paquetes recibidos de la capa de transporte en *datagramas* de internet, que incluyen un encabezado

y utiliza un algoritmo de ruteo para determinar si se envía el datagrama directamente o lo envía a algún ruteador. Finalmente pasa el datagrama a la interfase de red apropiada para su transmisión. Esta capa, también recibe datagramas desde la interfase de red, revisa su validez y utiliza el algoritmo de ruteo para decidir si el datagrama se procesa localmente o debe retransmitirse.

Capa de Interfase de Red. Esta interfase es responsable de aceptar datagramas IP y transmitirlos sobre una red específica. Puede consistir de un dispositivo o subsistema complejo que utiliza su propio protocolo de liga de datos.

1.4.2. Comunicación entre aplicaciones DICOM

DICOM establece especificaciones para sistemas que pretenden ser compatibles, definiendo *Entidades de Aplicación (AE-Aplication Entity)*. DICOM ofrece a las AE's tres alternativas de comunicación para el establecimiento de asociaciones. En las versiones anteriores, se hizo la especificación para comunicar aplicaciones punto a punto. DICOM versión 3.0, agrega la posibilidad de conexión en red utilizando como base los protocolos TCP/IP y los propuestos por el modelo de referencia ISO/OSI. Como se observa en la figura 1.5, DICOM aprovecha los protocolos definidos en las capas inferiores tanto de TCP/IP como de ISO/OSI y define los protocolos necesarios en las capas superiores (partes sombreadas de la figura 1.5) para soportar la comunicación entre AE's en forma eficiente.

En el caso de ISO/OSI, aprovecha los servicios de las primeras 6 capas, además de los elementos de servicio para el control de asociacion (ACSE). La otra alternativa de comunicación, es utilizar los protocolos de TCP/IP en donde se define el Protocolo de Capa Superior DICOM (*DUL-DICOM Upper Layer*), por encima de ellos. Para los tres casos, se definen los mismos servicios (servicios de capa superior OSI, Presentación+ACSE) en la última capa, los cuales permiten la portabilidad entre ambientes sin afectar a las aplicaciones DICOM.

El contexto de estudio de este trabajo es el protocolo DUL montado sobre TCP/IP. La decisión de utilizar TCP/IP obedece a que se ha implantado en prácticamente cualquier plataforma. Esto permite un uso amplio por diferentes aplicaciones. Al definir aplicaciones tipo PACS, utilizando DICOM, no es necesario invertir en software para comunicación en red. El protocolo DUL se describe brevemente a continuación.

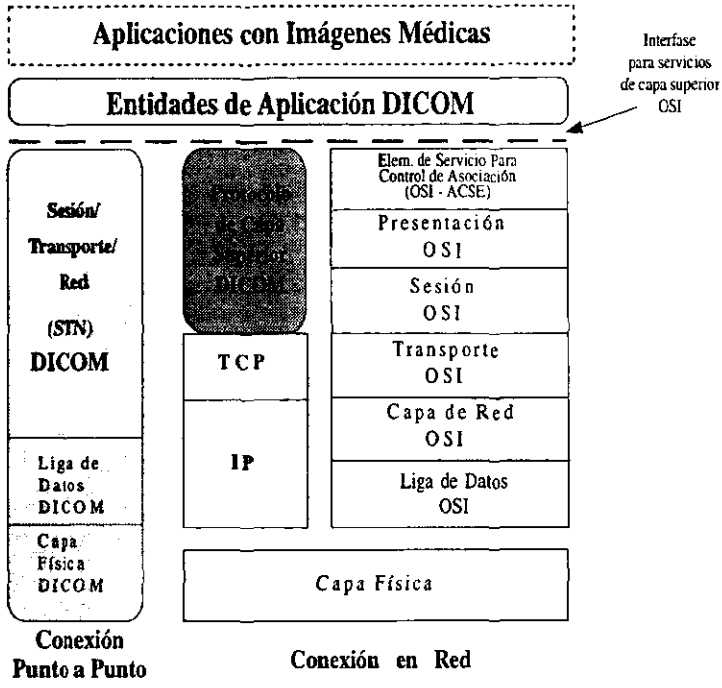


Figura 1.5: Comunicación en red para aplicaciones DICOM

1.4.3. El Protocolo de capa superior DICOM (DUL)

El protocolo DUL, comparado con el modelo de referencia ISO/OSI, sustituye a las capas de sesión, presentación y servicios de ACSE (parte más oscura de la figura 1.5) y se monta sobre el protocolo TCP para ofrecer a las aplicaciones DICOM comunicación en una red de computadoras.

El protocolo esta especificado básicamente a través de una máquina de estados finitos, en donde se contemplan 13 estados, 19 eventos y 28 acciones, que en conjunto ofrecen servicios para el establecimiento de asociación (A-ASSOCIATE), transferencia de información (P-DATA), liberación de asociación (A-RELEASE) y aborto de asociación (A-ABORT, A-P-ABORT). La utilización de estos servicios se realiza a través de primitivas de petición, indicación, respuesta y confirmación. Estos servicios son compatibles con los especificados para la comunicación a través de los protocolos de ISO/OSI y punto a punto para aplicaciones DICOM. De esta forma, las aplicaciones pueden utilizar cualquiera de estos medios de comunicación en forma transparente.

1.5. Objetivos de la tesis

Las especificaciones del estándar DICOM se han utilizado en varias implementaciones. Una de ellas, para propósito de demostración, contempla el desarrollo de una gran parte de las especificaciones de DICOM, incluyendo DUL, utilizando como herramienta el lenguaje de programación C [RSNA93]. Sin embargo, después de haber realizado un análisis a este producto, se observaron varios problemas al intentar adaptarlo a un sistema específico. Esto se debe principalmente a que su documentación no es muy completa y para realizar algún cambio, se debe invertir un tiempo significativo para entender la implementación.

En [Jensch98], se describe un software que implementa la mayor parte de las especificaciones de DICOM reutilizando código de [RSNA93]. Este software se ofrece en dominio público. El problema con él es que el código que se ofrece tiene algunas partes realizadas en lenguaje C y otras en C++, expresando en principio, una mezcla entre el modelo estructurado y el modelo orientado a objetos. En consecuencia, intentar entenderlo, se vuelve demasiado complicado.

Existe también el caso de las implementaciones comerciales, en donde el fabricante ofrece el software dentro de un paquete para armar un PACS, pero sin acceso a los programas fuente. Cabe señalar que dicho paquete, el cual incluye algunas estaciones de trabajo, servidores de impresión e imágenes y alguna aplicación, entre otras cosas, tiene un costo cercano al millón de dolares.

Considerando los aspectos anteriores, se ha planteado el desarrollo de las especificaciones del estándar DICOM. El producto sería una serie de bibliotecas que permitan la creación de AE's estandarizadas a un nivel de compatibilidad específico que formen parte de un sistema PACS. Para este desarrollo, se pretenden seguir los lineamientos de una de las *metodologías orientada a objetos*, la cual ofrece una serie de características de calidad (*robustez, flexibilidad, portabilidad, extensibilidad*, entre otras) acordes con la *Ingeniería de Software* y que ha motivado su utilización. Adicionalmente, existe una serie de herramientas para soportar la metodología que, en teoría, agilizan el proceso de desarrollo.

El proyecto como un todo es extenso, por lo que, después de analizar la información de las especificaciones [NEMA93], se dividió en tres partes principales: el protocolo DUL, los protocolos DIMSE (*DICOM Service Element*) para intercambio de mensajes y las entidades de aplicación DICOM.

Como se mencionó anteriormente, el contexto de este trabajo se ubica en el protocolo DUL y se plantean los objetivos siguientes:

1. Realizar el proceso de desarrollo de software utilizando el modelo orientado a objetos para el protocolo de capa superior DICOM (DUL) a partir de TCP/IP.

2. Trabajar con herramientas apropiadas para el desarrollo del proceso orientado a objetos

3. Sentar las bases para el desarrollo de sistemas PACS estandarizados en DICOM

1.6. Organización del documento

Este documento de tesis está estructurado en cinco capítulos. En el capítulo 2, se plantea la metodología y las herramientas utilizadas para el desarrollo del protocolo. El capítulo 3 describe las diferentes fases de la metodología aplicadas al desarrollo del protocolo, detallando las salidas obtenidas en cada etapa. En el capítulo 4, se hace un resumen y análisis de los resultados. Por último, en el capítulo 5, se plantean las conclusiones y las perspectivas que abre este trabajo.

2. METODOLOGÍA

2.1. Proceso de desarrollo de software

El proceso de desarrollo de software [Humphrey89] consiste en un conjunto de actividades en ingeniería de software, necesarias para transformar los requerimientos de un usuario en software. Un proceso de desarrollo de software tiene como finalidad establecer un nivel de calidad y control en el desarrollo del producto. El modelado de este proceso es la construcción de una descripción abstracta que lo represente.

A partir de los años 70, se han desarrollado diversos modelos del ciclo de vida del software, que han servido como mecanismos de evaluación y comparación, así como para mejorar la eficiencia en el desarrollo de software [Ramam96]. Además, han propiciado la generación de técnicas y herramientas para desarrollo, entrenamiento de personal y planeación completa de proyectos. Dentro de estos modelos, se encuentran el de cascada, el de espiral, el de prototipos y el incremental [Humphrey89].

El modelo de cascada, consiste en aplicar sucesivamente las fases de requerimientos, especificación, planeación, diseño, implementación e integración. Cada fase no se considera completa hasta que su documentación es terminada y sus productos son aprobados. Una vez que el cliente acepta el producto, cualquier cambio implica mantenimiento. Esto establece algunas desventajas: nunca se aplica a proyectos reales, es difícil establecer explícitamente todos los requerimientos desde el principio y las versiones de software son disponibles con retraso.

El modelo de espiral, consiste en aplicar cíclicamente cuatro fases: (1) determinación de objetivos, (2) evaluación de alternativas, identificación y solución de riesgos, (3) verificación del desarrollo y (4) planeación del siguiente ciclo. Tiene como ventajas poder identificar riesgos durante el proceso, verificar en forma continua el proceso y soportar el reuso de software existente. Sus principales desventajas se refieren a que se utiliza sólo para desarrollo interno de proyectos de software a gran escala y que consume mucho tiempo, lo cual implica retrasos en liberación.

El modelo de prototipos, consiste en elaborar diferentes partes del proceso, ya

sea para eliminar versiones anteriores o cambiarlas en forma evolutiva. La ventaja de usar este modelo es que permite verificar la exactitud de los requerimientos y la validez de los criterios de diseño para eliminar algunos riesgos. Sin embargo, este modelo no define en sí un proceso, por lo que debe ser utilizado en combinación con otros modelos, como el de cascada o el de espiral, en alguna de sus etapas.

El modelo incremental permite construir partes de la aplicación estableciendo el seguimiento del proceso por pasos, obteniendo retroalimentación fina en cada etapa y resultados parciales validados desde etapas tempranas del proceso. Aplicado en forma iterativa permite afinar estos resultados parciales. Dentro de sus ventajas, se tienen el manejo de riesgos y la posibilidad de liberar el producto gradualmente. Sus principales desventajas son el cambio contínuo de requerimientos y los problemas de integración al tratar de incorporar subproductos en la estructura existente, sin destruir lo que se construyó con anterioridad.

Considerando como base estos procesos, se ha observado una evolución natural de los procesos de desarrollo de software propiciado por la definición de dominios más complejos, que son la base de aplicaciones de gran tamaño [Ramam96]. De esta forma, han surgido procesos de desarrollo orientado a objetos que han incorporado los conceptos de modularidad, abstracción y reuso. El reuso se lleva a cabo utilizando componentes existentes (composición), aplicando herencia (especialización o extensión), adaptando patrones y arquitecturas en la fase de diseño y reutilizando código (paquetes y módulos).

Los sistemas de tiempo real y concurrentes han contribuido también a la evolución del proceso de desarrollo de software, al establecer requerimientos formales y verificación rigurosa para integrar vistas funcionales y de comportamiento del software, agregando características de seguridad y confiabilidad al ciclo de vida.

El proceso de software también se ha visto influenciado por organizaciones reguladoras y de certificación que buscan repetibilidad y predictibilidad en el proceso. De esta forma, se han propuesto reglas de productividad y calidad, a través de los criterios de ISO 9000 y del modelo de capacidad y madurez (*CMM-Capability Maturity Model*), propuesto por los institutos de ingeniería de software.

Por otro lado, existen varias herramientas CASE (*Computer-Aided Software Engineering*) que han automatizado partes del proceso de desarrollo, tales como generación de código, lenguajes de 4^a generación, modelado de requerimientos, especificaciones formales, verificación, programación visual y manejo de configuración. De igual manera, el desarrollo de sistemas basados en conocimiento ha contribuido a la evolución del proceso de desarrollo, al convertir requerimientos a código que utilizan transformaciones que preservan exactitud.

El proceso de desarrollo de software también se puede ver afectado por el

grado de experiencia del desarrollador. De esta forma, desarrolladores experimentados invierten más tiempo en las fases de requerimientos y diseño, que en modificaciones y depuraciones de código fuente, puesto que se ha observado mayor productividad al trabajar así.

2.2. Proceso de desarrollo orientado a objetos

Existen varias propuestas en cuanto al proceso de desarrollo Orientado a Objetos. En general, sus objetivos son:

- establecer reglas para el orden de actividades de un grupo de desarrollo,
- especificar los productos que deben ser desarrollados,
- dirigir el trabajo individual y de grupo de los desarrolladores y
- establecer criterios de inspección y medición a los productos asi como a las actividades definidas para un proyecto.

Entre las diferentes formas de desarrollar proyectos, se encuentra el manejo por arquitectura que se adapta en forma natural a proyectos orientados a objetos [Booch96], [Ambler97b] y [Booch97]. La arquitectura en esta metodología es expresada en dos formas. Por un lado, en un conjunto de clases organizadas en varias jerarquías y por el otro, en la forma en la que las clases cooperan para proveer la funcionalidad de un sistema.

El proceso de desarrollo de software utilizado para este trabajo está especificado en [Booch96], en donde se establece que para obtener una arquitectura adecuada del sistema se aplica el proceso iterativo incremental. Este proceso es iterativo porque envuelve refinamientos sucesivos de la arquitectura del sistema, aplicando la experiencia y los resultados principales de la iteración anterior en el proceso de análisis y diseño de la siguiente iteración. El proceso es incremental porque cada paso de análisis, diseño e implementación nos lleva a:

- refinar gradualmente las decisiones sobre la arquitectura y las herramientas particulares del sistema,
- extender la visión de la arquitectura inicial y
- obtener un producto de software terminado.

El proceso de desarrollo se subdivide a su vez en dos subprocesos: el macroproceso y el microproceso. En el macroproceso de un proyecto orientado a objetos,

se deben contemplar los refinamientos sucesivos de la arquitectura del sistema, además de que cada resultado parcial debe generarse considerando riesgos. Esta etapa sirve para controlar las actividades del proyecto en forma global e involucra actividades de descubrimiento, invención e implementación. El microproceso se aplica simultáneamente a estas actividades y consiste en trabajar a niveles de abstracción más bajos y específicos del problema a resolver. Las actividades del microproceso pueden llevar a modificar el enfoque global si el análisis de los riesgos así lo indica.

En las siguientes secciones, se analizan en detalle las etapas de cada uno de estos subprocesos.

2.2.1. Macroproceso

2.2.1.1. Conceptualización

Consiste en establecer todos los requerimientos para un nuevo sistema o para los cambios importantes de uno ya existente. En esta etapa, se deben identificar los riesgos importantes en el proyecto. Por ejemplo: analizar las repercusiones en el uso de nueva tecnología (nueva plataforma, nuevos lenguajes, nuevas herramientas, etc) y tomar decisiones apropiadas para su manejo. La conceptualización se realiza conjuntamente entre el cliente y el desarrollador, lo cual permite también establecer un vínculo de colaboración. Se recomienda el desarrollo de prototipos ejecutables para la correcta definición de los requerimientos. Los resultados principales de esta etapa son el conjunto de requerimientos y de riesgos y, en su caso, el prototipo ejecutable.

2.2.1.2. Análisis

Es una etapa crucial en el macroproceso la cual consiste en desarrollar un modelo de comportamiento del sistema. El análisis debe llevar a una descripción detallada del funcionamiento del sistema con un énfasis en el comportamiento (qué hace) y no en la forma (cómo lo hace). Para el correcto desarrollo de esta etapa, es importante definir:

- Las funciones principales del sistema.
- El comportamiento que el sistema asume para realizar cada una de estas funciones.
- Las variantes en este comportamiento.

- Las responsabilidades de las clases y objetos que contribuyen a este comportamiento.
- La interacción del sistema con otros.
- Los riesgos que no hayan sido determinados en la etapa de conceptualización.

Al considerar únicamente el comportamiento, es posible identificar las formas de interacción ("function points") que realiza el sistema visto como una *caja negra*, es decir, la actividad del mismo en respuesta a un evento. Estas formas de interacción se pueden agrupar en *casos de uso*. El número de formas de interacción que se pueden definir permiten determinar la complejidad y cada forma de interacción se puede traducir en un escenario para un sistema en desarrollo.

Los principales productos obtenidos en esta etapa son:

- Una descripción del *contexto del sistema.*
- Un conjunto de escenarios que describen en forma completa su funcionamiento, o *modelo de comportamiento.*
- Un *modelo de dominio* que describe los aspectos centrales del problema del mundo real.
- Un conjunto más completo de los riesgos.

Para la obtención de estos subproductos puede también ser necesario el desarrollo de un prototipo ejecutable.

En la etapa de análisis, es necesario definir varias herramientas que faciliten la descripción del comportamiento del sistema a diferentes niveles de abstracción. A un nivel muy global, se definen los *casos de uso* que especifican un aspecto o faceta del comportamiento del sistema. Asociado a un caso de uso, se tiene un conjunto de escenarios (*instancias del caso de uso*), cada uno de los cuales describe una forma de interacción única. Es importante hacer notar que un escenario no permite definir la arquitectura del sistema, sino ilustrar las interacciones dinámicas que se dan en el contexto de cierta arquitectura.

De los casos de uso y escenarios, se obtiene un conjunto de clases, responsabilidades y colaboraciones que se modelan con ayuda de tarjetas CRC (*Class Responsability Colaborators*); éstas serán descritas en la sección 2.3. El modelo de dominio es un esquema obtenido a partir de las clases identificadas en el vocabulario del problema y el tipo de interaccion entre ellas.

2.2.1.3. Diseño

El objetivo de esta etapa es proponer una arquitectura flexible que pueda incluir nuevos requerimientos, dando énfasis a la estructura tanto estática como dinámica, del sistema en desarrollo. Para lograr este objetivo, se debe definir la estructura básica del sistema, identificar patrones que se adapten y simplifiquen esta estructura, identificar las partes del sistema que puedan ser reutilizadas o adaptadas y definir una estrategia que permita extender la arquitectura propuesta a la etapa de producción.

Como resultados de esta etapa se debe esperar:

- Una arquitectura ejecutable.
- Una especificación de los patrones identificados.
- Un plan de producción.
- Criterios de prueba.
- Un conjunto actualizado de los riesgos, si existen.

En algunos casos, se puede requerir modificar la arquitectura propuesta en una etapa posterior de evolución del sistema.

2.2.1.4. Evolución

Consiste en realizar la representación de la arquitectura obtenida en la etapa de diseño empleando un lenguaje adecuado y refinarla iterativamente, hasta que se considere que el sistema es acorde con los requerimientos planteados. Asimismo, se deben comprobar los requerimientos reales del sistema a través de pruebas de comporatmiento cuya realización se planeó para esta fase. Es recomendable intentar comprobar la flexibilidad del sistema proponiendo nuevos requerimientos y comprobando la simplicidad de su realización. Es en esta etapa en la que se aplica fuertemente el microproceso.

2.2.1.5. Mantenimiento

El proposito de esta etapa es manejar requerimientos posteriores a la liberación del sistema. Los cambios en el tiempo son inherentes a todo sistema de software y se plantea que los sistemas manejados por arquitectura, bien desarrollados, deben responder a estos cambios. En esta etapa, se comprueba si en realidad la arquitectura es lo suficientemente robusta para facilitar la realizacion de nuevos requerimientos.

2.2.2. Microproceso

El microproceso se caracteriza por 4 principales atributos:

1. Es cíclico en su aplicación a cada parte del sistema o al observar resultados a diferentes niveles de abstracción.

2. Es oportuno porque cada ciclo parte de lo que ya se conoce y se tiene la oportunidad de refinar el trabajo en cada paso subsecuente.

3. Su enfoque es basado en roles y responsabilidades en lugar de funciones y control.

4. Es pragmático porque se apega a aspectos reales.

El microproceso se aplica principalmente en las etapas de análisis, diseño y evolución del macroproceso, adaptandose a los sus objetivos particulares. Las fases que componen el microproceso se describen en las siguientes secciones.

2.2.2.1. Identificación de clases y objetos

Su propósito es seleccionar abstracciones que modelen correctamente el problema en cuestión. Se aplica en diferentes contextos del sistema como son: el dominio del problema y las diferentes interfases que se demanden. El principal producto de este paso es un diccionario de abstracciones que sirve como una fuente de información central para el vocabulario del dominio del problema y de su solución. Las abstracciones pueden registrarse utilizando como herramienta tarjetas CRC. Durante el proceso de desarrollo, este diccionario crecerá y cambiará de acuerdo a la profundidad con la que se esté trabajando el problema y su solución. El diccionario será refinado agregando nuevas abstracciones, eliminando algunas que sean irrelevantes y consolidando otras. Algunas de las entradas al diccionario podrían llegar a ser clases, otras objetos y otras simplemente atributos o sinónimos de otras abstracciones. Si se utilizan herramientas CASE en el proceso de desarrollo, es posible que el manejo del diccionario se realice en forma automática.

La principal actividad relacionada con este paso es el descubrimiento y la invención de abstracciones. Un orden de eventos típico para estas actividades es:

- Generar un conjunto de candidatos a abstracciones, buscando entidades con propiedades similares. Las cosas tangibles son un buen punto de partida. Después, tratando de rastrear eventos externos, se encontrarán más abstracciones a otros niveles, observando que por cada evento deben existir objetos responsables de detectar y/o reaccionar a él.

- Un sistema posee comportamiento observable hacia el exterior. De este comportamientos, se deben tratar de encontrar los roles y responsabilidades de las entidades del dominio que los realizan, representando estos a las abstracciones participantes.

- Revisar los escenarios que describen el comportamiento a altos niveles de abstracción. Considerar después los escenarios que describen mayor detalle, así como los escenarios secundarios que describen las *partes oscuras* del comportamiento deseado por el sistema.

Considerar que algunas abstracciones identificadas al principio pueden ser erróneas, lo cual no representa una equivocación, sino parte del proceso. Cuando se aprende más acerca del problema, se pueden tomar decisiones de cambiar las fronteras de ciertas abstracciones. Esto se logra reasignando responsabilidades, combinando abstracciones similares y/o dividiendo grandes abstracciones en grupos de colaboradoras. De esta forma, se establece un nuevo modelo de solución.

2.2.2.2. Identificación de la semántica de clases y objetos

Tiene como objetivo determinar una distribución apropiada de responsabilidades entre clases y objetos identificados en las etapas del proceso. Esto involucra la identificación de roles y responsabilidades para las partes que componen el modelo de dominio o de las diferentes capas del sistema, asi como establecer una clara separación entre abstracciones con semántica claramente relacionada, agrupándolas juntas. En esta fase, el enfoque principal es hacia el comportamiento, considerando la representación como un aspecto secundario (es más importante el *que hace* que el *como lo hace*). Los resultado que deben obtenerse en esta fase comprenden una especificación de roles y responsabilidades de las abstracciones clave y software que codifica estas abstracciones (a nivel interfase). Las diferentes representaciones a utilizar dependen de la etapa en la que se encuentre el proceso de desarrollo. De esta forma, se pueden emplear diagramas de escenarios, diagramas de clases o diagramas de estados.

El producto más importante en esta etapa son los roles y responsabilidades especificados. Estos se puede expresar en varias formas: desde la descripción en texto hasta la utilización de herramientas como tarjetas CRC. Los roles y responsabilidades serán transformados en métodos o funciones, cuando el modelo se encuentre en una etapa de refinamiento óptimo.

Las reglas generales para la realización de esta fase son:

- Identificar patrones de interacción entre abstracciones a partir del conjunto de escenarios obtenido.

- Identificar patrones de comportamiento del conjunto de responsabilidades identificadas.

- Buscar patrones dentro de operaciones particulares, cuando estas ya se encuentren definidas en las últimas fases del macroproceso.

2.2.2.3. Identificación de las relaciones entre clases y objetos

En esta fase, se plantean un conjunto de mecanismos simples que regulan las clases y los objetos que hasta ese momento se hayan identificado en el proceso de desarrollo. La actividad principal es la invención, que involucra decisiones sobre la dependencia semántica entre cada clase y objeto, así como entre grupos de ellos. De esta forma, se debe lograr que las relaciones entre clases y entre objetos complementen las fronteras conceptuales de cada abstracción.

Se recomienda que se intente descubrir los patrones comunes de relación y se invente la forma de explotar estos patrones. También, maximizar las conexiones entre abstracciones que son semánticamente relacionadas y minimizarlas cuando se tiene cierta distancia semántica que esta sujeta a cambio.

Dentro de los resultados que se deben obtener, se tienen la especificacion de las relaciones entre abstracciones clave, software que represente estas especificaciones, así como diagramas que expresen el significado de las relaciones y las colaboraciones importantes.

Las reglas generales para la realización de esta fase son:

- Considerar a las clases que se encuentran en el mismo nivel de abstracción o que se identifican con un grupo de escenarios particular. Dar semántica a estas abstracciones asignándoles operaciones y atributos necesarias para expresar las propiedades importantes del problema que está siendo modelado.

- Buscar relaciones de asociación entre las clases consideradas, buscando dependencias semánticas entre pares de ellas.

- En cada asociación encontrada, establecer los roles de los participantes, así como la cardinalidad relevante entre ellas u otro tipo de restricción.

- Validar estas relaciones a través de escenarios que expresen la interacción entre objetos y aseguren que las asociaciones representen el comportamiento requerido.

- Después de refinar las relaciones de asociación, considerar relaciones de herencia, agregación, instanciación y uso[1].

2.2.2.4. Implementación de clases y objetos

El objetivo de esta fase es representar cada abstracción y sus mecanismos en la forma más eficiente y elegante. Esto significa obtener abstracciones prácticas. El resultado de la implementación es el software que representa a las clases y sus mecanismos.

La actividad principal corresponde a la selección de estructuras y algoritmos apropiados que complementen los roles y responsabilidades de todas las abstracciones identificadas en las fases anteriores del microproceso. De esta forma, las tres primeras fases del microproceso se enfocan a la vista externa de las abstracciones y esta última a la vista interna.

Las reglas generales para la realización de esta fase son:

- Para cada clase o colaborador de ésta, identificar los patrones de uso entre sus clientes para determinar las operaciones centrales que puedan ser optimizadas.
- Antes de considerar una representación desde cero, considerar la adaptación de clases existentes, por herencia o por instanciación.
- Considerar objetos a los que se les pueda delegar responsabilidades.
- Si la semántica de una abstracción no se puede lograr a través de la herencia, instanciación o delegación, se debe considerar una representación apropiada utilizando las primitivas del lenguaje, recordando siempre lo que sus clientes esperan de ella y realizando una representación eficiente para los patrones de uso esperado.
- Seleccionar algoritmos apropiados para cada operación. En operaciones complejas, dividir el problema en varias funciones menos complejas que puedan reutilizarse. Considerar también las ventajas y desventajas de calcular ciertos estados de una abstracción o almacenarlos.

2.3. Herramientas y técnicas

A través del tiempo, se han definido una serie de herramientas y técnicas que ofrecen soporte al proceso de desarrollo orientado a objetos. Varias de ellas se

[1]Las clases no pueden existir en forma aislada. Por lo tanto, en [Booch94] se definen las diferentes formas en las que las clases se pueden relacionar.

han estandarizado y otras por su gran utilidad han prevalecido en el tiempo. Las herramientas y técnicas utilizadas en este trabajo se describen enseguida.

2.3.1. Tarjetas CRC (*Class Responsability Colaborators*)

Las tarjetas CRC, propuestas por Kent Beck y Ward Cunningham en 1989, son utilizadas para modelar el proceso de identificar los requerimientos del usuario en una aplicación orientada a objetos [Ambler97a]. Esta herramienta se ha utilizado, para que, en forma sencilla, se visualicen escenarios [Booch96]; su sencillez radica en la facilidad con la que pueden manipularse. Se trata de unas cuantas docenas de tarjetas indexadas, lápiz y goma, que se utililizan para describir abstracciones, sus responsabilidades y sus colaboradores, a través del lenguaje natural. Las tarjetas sirven para promover una interacción activa entre el grupo de desarrollo y expertos de dominio con la finalidad de plantear y entender los requerimientos, comportamientos e impacto del sistema, forzando a considerar una distribución inteligente de responsabilidades hacia la arquitectura.

En [Booch96] se sugiere utilizarlas durante la fase de análisis y en las últimas etapas de diseño y evolución, para analizar *aspectos oscuros del sistema.* Sin embargo, la principal limitación de las tarjetas es que no comunican los aspectos dinámicos en un escenario, tales como la interacción explícita entre objetos o el comportamiento de objetos a través del tiempo.

Algunas reglas propuestas por [Ambler97a] para modelar con esta herramienta son:

- Encontrar clases
- Encontrar responsabilidades
- Definir colaboradores
- Establecer casos de uso
- Reordenar las tarjetas

Cabe señalar que existen varias herramientas CASE que utilizan las ideas de las tarjetas CRC, para realizar el modelado a través de una computadora.

2.3.2. El lenguaje de modelado UML (*Unified Modeling Language*)

UML representa a un conjunto de notaciones para análisis y diseño orientado a objetos. Fue desarrollado por Grady Booch, James Rumbaugh e Ivar Jacobson, en colaboración con varias compañías que representan a la industria relacionada al desarrollo de software.

La notación empleada por UML es independiente de cualquier lenguaje de implementación y representa la convergencia de ideas en diagramas de análisis y diseño orientados a objetos, principalmente propuestos por los autores. De esta forma, se plantea como un lenguaje común en la comunicación de ideas entre desarrolladores de software. Actualmente, la OMG (*Object Management Group*) lo ha aprobado como lenguaje de modelado estándar.

UML ofrece notación para modelar a través de diagramas que expresan vistas lógica, física, estática y dinámica [Booch94]. Desde otra perspectiva, en [Booch97] se clasifican como diagramas de estructura estática, diagramas de caso de uso, diagramas de secuencia, diagramas de colaboración, diagramas de estado de clases, diagramas de actividad y diagramas de implementación. Sus descripciones amplias y notación se pueden consultar en [Booch97], de los cuales se realiza una breve descripción a partir de [Bergner97].

Diagramas de estructuras estáticas. Modelan datos en un sistema orientado a objetos y pueden contener información de la funcionalidad de los componentes de estos datos. Los diagramas de estructura estática tienen dos variantes: *Diagramas de Clases* y *Diagramas de Objetos*. Los primeros representan a las clases que forman parte de un sistema o una porción de éste, sus atributos, operaciones, relaciones y dependencias entre ellas. Los segundos representan objetos y sus relaciones, formando gráficas que pueden extenderse durante el tiempo de ejecución de un sistema. Estos diagramas son utilizados para modelar abstracciones en las primeras fases de desarrollo y posteriormente son enriquecidos con atributos y operaciones adicionales para que, finalmente, sean traducidos en esqueletos de clases.

Casos de uso. Se utilizan para modelar *actores externos* que requieren de algún tipo de interacción, a muy alto nivel de abstracción, con el sistema propuesto a través de sus diferentes casos de uso. Un actor puede representar a un usuario, organización u otro sistema, externos al sistema en estudio. Asociado a un caso de uso, se tiene un conjunto de escenarios (o instancias del caso de uso) que describen una forma de interacción única o una funcionalidad particular del sistema.

Diagramas de secuencia. Conocidos también como *diagramas de trazo de mensajes*. Muestran la interacción entre objetos a través de envío de mensajes en una secuencia de tiempo. Los diagramas de secuencia modelan sucesiones explícitas de mensajes y su uso principal es para modelar la interacción de objetos en tiempo de ejecución y escenarios complejos.

Diagramas de colaboración. Son una forma especial de diagramas de objetos enriquecidos con información relacionada al flujo de mensajes entre objetos, así como su creación y destrucción. Aunque la notación para los diagramas de colaboración es diferente a los diagramas de secuencia, ambos contienen casi la misma información. La diferencia principal es que los diagramas de secuencia tienen su enfoque en el orden temporal de eventos, mientras que los diagramas de colaboración se concentran en las relaciones y conexiones entre objetos.

Diagramas de estados. Los objetos obtenidos a partir de una clase se crean con ciertos valores iniciales de sus atributos, representando un estado inicial. Sin embargo, con el paso del tiempo, sus atributos pueden cambiar de valor. Los diagramas de estados modelan el estado de los objetos durante su ciclo de vida, considerando acciones (codificadas a través de sus métodos) como respuesta a eventos (invocación a sus métodos).

Diagramas de actividad. Son un caso especial de los diagramas de estado. Para estos diagramas, todos los estados son estados-acciones, de tal forma que, todas las transiciones son disparadas al completarse las acciones de los estados origen. Los diagramas de actividad pueden modelar clases, la implementación de una operación o casos de uso. Su objetivo es expresar el flujo manejado por un procesamiento interno. Son utilizados regularmente en la ocurrencia de eventos asíncronos.

Diagramas de implementación. Se proponen en dos variantes. *Diagramas de componentes*, que muestran la estructura del código fuente y su partición en componentes y *Diagramas de procesos* ("deployment"), que modelan la implementación de la estructura en tiempo de ejecución y la distribución de objetos y componentes en nodos físicos de computadoras.

2.3.3. Patrones de diseño

Su origen en el área de computación se da a partir de patrones arquitectónicos propuestos por el arquitecto Christopher Alexander [Lea94] [Khare95] [Coplien94]. En los años 60's, los arquitectos comenzaron a explorar el diseño arquitectónico

automatizado a través de computadoras, lo cual resultó en sistemas de construcción modular. Sin embargo, tiempo después, Christopher Alexander rompió con este movimiento al notar que las grandes arquitecturas de la historia no se realizaron con base a diseños rigurosos y planeados. Sin embargo, guardaban cierta armonia con su rededor. También, observó que algunos edificios eran estéticamente más agradables que otros y que la estética fue frecuentemente armonizada con necesidades humanas y de confort. Encontró temas repetitivos en arquitectura, capturándolos en descripciones e instrucciones (a lo que le llamó *Patrón*) que relaciona a la réplica con similitud en diseños.

Alexander estableció que cada patrón debe contemplar idealmente ciertas propiedades como son [Lea94]:

1. **Encapsulamiento**, cada patrón representa a un par problema/solución bien definidos.

2. **Generalidad**, cada patrón debe contener un proceso auto-establecido para poderlo realizar, que lo hace claro y entendible.

3. **Equilibrio**, cada patrón provee una razón para cada paso de diseño, adaptable a situaciones particulares.

4. **Abstracción**, los patrones representan abstracciones de experiencia empírica y conocimiento cotidiano en un contexto específico, aunque no necesariamente universal.

5. **Adaptabilidad**, los patrones pueden ser adaptados a niveles finos de detalle o a niveles mayores.

6. **Composición**, los patrones se relacionan en forma jerárquica.

También, estableció que los patrones deben incluir breves reglas de como aplicarlos y relacionarlos.

Desde la década pasada, los diseñadores de software han descubierto analogías entre los patrones de Alexander y los patrones en las arquitectura de software. Esto ha llevado a implantar patrones utilizando la metodología orientada a objetos. De hecho, los patrones de Alexander establencen una relación directa a construcciones orientadas a objetos. Las clases se expresan a través de una vista externa y otra interna [Booch94]. En la vista externa, se define la interfase de la clase para especificar sus atributos y responsabilidades. A través de las responsabilidades se definen los servicios para clientes externos. En la vista interna, por otro lado, se expresa la solución a las especificaciones de la vista externa. Los

patrones pueden plantearse de la misma manera, de tal forma que las clases bien especificadas comparten las propiedades mencionadas anteriormente para los patrones. Sin embargo, como los patrones pueden describir conceptos y estructuras sin ser objetos, el término patrón puede considerarse como más conveniente. De esta forma, los patrones incrementan la expresión y el nivel de descripción soportado por estructuras apegadas al modelo orientado a objetos y, a su vez, los conceptos del modelo pueden aplicarse para reforzar los diseños basados en patrones.

Por otro lado, en [Coplien97], se consideran varios aspectos que le dan mayor valor a los patrones al establecer una visión más amplia de una aplicación. Dentro de estos aspectos está el considerar a los patrones más allá del modelo orientado a objetos, más allá de la arquitectura de software y más allá de factores humanos. Los patrones complementan a métodos de diseño existentes, resolviendo problemas que se salen de su contexto. A su vez, representan abstracciones de alto nivel en herramientas de desarrollo (o "Frame-works") que, en algunos casos, hacen mezclas de paradigmas y hacen evidente como el modelo orientado a objetos es superado por los patrones. En cuanto a arquitectura de software, es interesante observar como los patrones estructurales y sus relaciones ayudan a definirla. Sin embargo, actualmente los patrones son utilizados además en enseñanza, organizaciones y procesos [Coplien94], superando así a las arquitecturas de software. Al hablar de sistemas de software, debe considerarse las facetas que intervienen en su desarrollo que van desde lo tecnológico hasta lo humanístico, estableciendo una combinación para una arquitectura con componentes estructurales y humanos.

Un diseñador de software, familiarizado con un conjunto de estructuras de diseño, puede aplicar dichas estructuras inmediatamente en un nuevo diseño sin tener que redescubrirlas. De esta forma, las estructuras de diseño facilitan el *reuso de arquitecturas exitosas*, expresando técnicas ya probadas que se pueden compartir con nuevos diseñadores para nuevos sistemas.

Tradicionalmente, al trabajar en el desarrollo de un sistema de software, la complejidad es atacada dividiendo el sistema en varias partes aisladas y poniendo atención en la estructura interna e interfase externa de cada parte. Las interfases capturan el comportamiento formal del sistema, pero el comportamiento del sistema como un todo es más rico en información que la suma de sus partes. La arquitectura captura y articula la imagen del sistema, sus relaciones y sus partes. *La expresión del diseño en terminos de relaciones entre partes de un sistema y las reglas para transformar esas relaciones es a lo que se le llama patrones de diseño* [Coplien97].

Los patrones de diseño representan los temas abstractos comunes en diseño orientado a objetos. De esta forma, través de un nombre (identificador), preser-

van la información de diseño al capturar la intensión que existe detrás de ellos. Específicamente, identifican clases, instancias, sus reglas, colaboraciones y la distribución de responsabilidades.

Los usos que se les puede dar a los patrones de diseño en el proceso de desarrollo orientado a objetos son [Gamma93]:

- **Un vocabulario común** para que los diseñadores se comuniquen, documenten y exploren alternativas de diseño, reduciendo así la complejidad de un sistema.
- **Constituyen una base de experiencia** para el desarrollo de software reusable. Actúan como bloques para la construcción de diseños más complejos. Pueden considerarse como *micro-arquitecturas* que contribuyen a la arquitectura completa de un sistema.
- **Ayudan a reducir el tiempo de aprendizaje** de una biblioteca de clases. Una vez que se han visualizado los patrones de cierta biblioteca, se puede reusar esta experiencia para aprender una nueva.
- **Ofrecen los medios para reorganizar o reconstruir jerarquías de clases** durante la etapa de diseño del ciclo de vida del software.

Los patrones de diseño consisten basicamente de 3 partes:

1. Una descripción abstracta de una colaboración de clases u objetos y su estructura. Se considera abstracta porque involucra un diseño abstracto y no un diseño particular.
2. El punto o problema considerado dentro de la estructura abstracta de un sistema. Esto define las condiciones en las que el patrón de diseño se puede aplicar.
3. Las consecuencias de aplicar la estructura abstracta a la arquitectura de un sistema. Esto determina la aplicabilidad del patrón, de acuerdo a las restricciones de diseño.

Los patrones de diseño son definidos en terminos del modelo orientado a objetos. Se consideran lo suficientemente abstractos para evitar la especificación de detalles de implantación, lo cual asegura una amplia aplicabilidad. Sin embargo, pueden ofrecer sugerencias para implantaciones potenciales.

2.4. Conclusión y discusión

El plantear como objetivos la utilización de un proceso de desarrollo de software orientado a objetos, así como de herramientas apropiadas tipo CASE, ha motivado el desarrollo de este capítulo.

En la sección 2.1, se planteó el proceso de desarrollo de software, con la finalidad identificar con precisión la ubicación de los procesos de desarrollo orientados a objetos.

El ejercicio de escoger la forma de atacar el problema a veces se vuelve tedioso por la falta de experiencia. Sin embargo, gracias a la discusión planteada en [Monarchi92] para las metodologías en análisis y diseño orientado a objetos, se hace evidente la propuesta de Booch como una de las más completas. La propuesta de Booch fue mejorada posteriormente en su publicación [Booch96], en donde se agregan aspectos importantes para el control del proceso de desarrollo de software a través del macroproceso. Algunos de estos aspectos son: ofrecer una visión global de la arquitectura del sistema en desarrollo, que trae como consecuencia un mejor control sobre los riesgos. Aunado a esto, la experimentación con herramientas CASE para automatizar el proceso de desarrollo fortaleció la elección. Dos de ellas, *Object-Domain* y *Rational Rose*, soportan desde sus primeras versiones la metodología de Booch, además de la de Coad y Yourdon (discutida en [Coad92]) para Objet Domain y OMT (Object Modeling Technique) para Rational Rose.

De esta forma, en la sección 2.2, se describe el proceso de desarrollo elegido y utilizado en este trabajo, planteado a través del macroproceso y del microproceso.

En la sección 2.3, se hace una descripción de las técnicas y herramientas que en [Booch96] se proponen como soporte al proceso de desarrollo en sus diferentes etapas.

Sin embargo, en la actualidad, se ha realizado una gran labor por parte de Grady Booch, James Rumbaugh e Ivar Jacobson, para estandarizar la notación a través del lenguaje de modelado UML y han realizado una propuesta a un proceso de desarrollo de software unificado. La conclusión de estos trabajos se publicó recientemente ([Booch99], [Rumbaugh99] y [Jacobson99]), en donde se hace más explícito el uso de la notación, así como las actividades a realizar en cada fase del proceso de desarrollo y los resultados concretos a obtener. Esto, en principio, facilitará aún más el proceso de desarrollo de software orientado a objetos.

3. Desarrollo del Protocolo de Capa Superior DICOM (DUL)

Para propósitos de claridad, la documentación que se presenta enseguida se establece en forma secuencial. Sin embargo, las actividades realizadas en el desarrollo del protocolo no siguió esta secuencia, sino lo planteado en [Booch96] para el proceso de desarrollo de software iterativo e incremental, como corresponde al uso de tecnología orientada a objetos.

3.1. Conceptualización

Desde el punto de vista del modelo de capas, un protocolo de comunicación es una entidad que utiliza los servicios de la capa inmediata inferior y ofrece sus servicios hacia la capa inmediata superior a través de primitivas. Para este desarrollo, DUL utiliza los servicios de TCP/IP y ofrece sus servicios a la capa de aplicación DICOM (ver sección 1.4.2). Esto nos lleva a identificar tres entidades importantes relacionadas con el protocolo DUL: una interfase hacia entidades de aplicación, otra interfase para usar los servicios de TCP/IP y la del protocolo DUL. En los siguientes párrafos se hace una descripción de cada una de ellas.

3.1.1. Interfase hacia entidades de aplicación

Para esta interfase, lo más importante es la forma en la que DUL ofrece sus servicios. DICOM especifica los requerimientos de comunicación para *entidades de aplicación (AE)*. Estos requerimientos son especificados de acuerdo a los servicios ofrecidos a través de primitivas, utilizando como base las especificaciones de ACSE (*Association Control Service Element*) y las especificaciones de la capa de presentación OSI. Esta interfase es común a las tres formas de comunicación especificadas para DICOM (ver figura 1.5). En la figura 3.1, se observa la forma en que las entidades de aplicación hacen uso de los servicios del protocolo DUL utilizando primitivas de *petición*, *indicación*, *respuesta* y *confirmación*. Cabe señalar que los servicios DUL se ofrecen a través de entidades *Proveedoras del Servicio DUL (PSDUL)*.

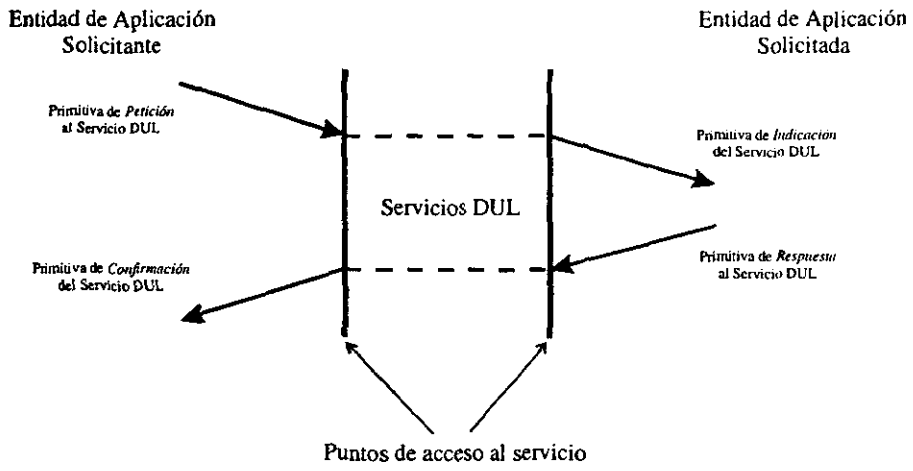


Figura 3.1: Entidades de Aplicación usando los servicios de DUL

Los servicios especificados por el protocolo son ([NEMA93] tomo 8): A-ASSOCIATE, A-RELEASE, A-ABORT, A-P-ABORT, P-DATA. La letra "A" que le antecede a cada servicio significa que éste es ofrecido por ACSE y la "P" significa que el servicio es ofrecido por la capa de presentación.

Los servicios pueden ser confirmados o no-confirmados. Para el primer caso, se sigue estrictamente el esquema de la figura 3.1 utilizando las 4 primitivas. Para el segundo caso, se sigue un mecanismo unidireccional utilizando sólo 2 primitivas. Cabe señalar que, en cada servicio deben intervenir dos AE's con su PSDUL respectivo.

Asociado a cada servicio se especifican una serie de datos considerados como parámetros del servicio. Algunos de los parámetros son utilizados por el servicio y otros están relacionados a información de las AE's. También, algunos de ellos son de uso obligatorio y otros son opcionales. Se trata básicamente de un subconjunto de los parámetros especificados por OSI-ACSE para cada servicio (se pueden consultar en [NEMA93] tomo 8, pags. 12-22).

3.1.1.1. Servicio A-ASSOCIATE

Este servicio es especificado como *confirmado* y es utilizado para el establecimiento de asociación entre dos AE's. Una AE DICOM que desea establecer una asociación (solicitante) debe emitir una *primitiva de petición* para el servicio A-ASSOCIATE. La AE solicitada es identificada a través de los parámetros de

la primitiva de petición. La AE solicitante no podrá enviar alguna otra primitiva, excepto la *primitiva de petición* para el servicio A-ABORT, hasta que haya recibido una *primitiva de confirmación* para el servicio A-ASSOCIATE. La AE solicitada recibirá la petición, a través de su PSDUL, representada por una *primitiva de indicación* para el servicio A-ASSOCIATE. Esta AE rechazará o aceptará la asociación emitiendo una *primitiva de respuesta* para A-ASSOCIATE, junto con un parámetro apropiado de resultado. La AE solicitante recibirá la respuesta, a través de su PSDUL, representada por una *primitiva de confirmación* para el servicio A-ASSOCIATE, conteniendo el mismo parámetro de resultado.

Si la AE solicitada acepta la asociación, éste canal de comunicación se utilizará por ambas AE's para los servicios especificados en el *contexto de aplicación* DICOM activo (especificado en los parámetros durante la asociación), con la excepción de A-ASSOCIATE. Esto implica que las AE's involucradas pueden intercambiar mensajes, especificados para la capa del protocolo de aplicación.

Si la AE solicitada rechaza la asociación, se romperá ca comunicación entre las dos AE's (solicitante y solicitada). Una de las causas del rechazo puede suceder cuando el PSDUL de la AE solicitada, no soporte la asociación requerida, en tal caso, regresará una primitiva de confirmación a la AE solicitante con el parámetro de resultado en un valor de rechazado. Al parámetro *Fuente de resultado* se le asignará un valor simbólico, relativo a que *la interfase del servicio de capa superior emitió el resultado.* En este caso, la primitiva de indicación no será enviada a la AE solicitada.

Dentro de los parámetros más importantes establecidos para el servicio de asociación, se encuentran:

- El nombre de un *contexto de aplicación*, que define explícitamente a un conjunto de elementos de servicio de la aplicación propuesta por el solicitante. Adicionalmente, contiene información necesaria para la conectividad de entidades de aplicación en una asociación.

- Uno o más *contextos de presentación*, que definen la presentación de información en una asociación. Un contexto de presentación está compuesto por una identificación del contexto de presentación, el nombre de una *sintaxis abstracta* y uno o más nombres de *sintaxis de transferencia.*

  - La *sintaxis abstracta* especifica a los elementos de datos en la capa de aplicación, con una semántica asociada o información de control para el protocolo de la capa de aplicación.
  - La *sintaxis de transferencia* define al conjunto de reglas capaces de representar, sin ambigüedad, a los elementos de datos definidos por

una o más sintaxis abstractas. Para el caso de DICOM, la negociación de sintaxis de transferencia permite, a las entidades de aplicación involucradas, establecer un acuerdo en las técnicas de codificación que son capaces de soportar. Por ejemplo: orden de bytes o compresión.

3.1.1.2. Servicio A-RELEASE

Este es un *servicio confirmado* utilizado para liberar en forma *normal* una asociación.

Una AE que desee liberar una asociación debe emitir una *primitiva de petición* para el servicio A-RELEASE. Una vez emitida, la AE no podrá enviar alguna otra primitiva excepto la de *petición* para el servicio A-ABORT, hasta recibir una *primitiva de confirmación* para el servicio A-RELEASE.

El PSDUL de la AE solicitada emitirá una *primitiva de indicación* para el servicio A-RELEASE y la AE solicitada podrá contestar con: una *primitiva de respuesta* para el servicio A-RELEASE, una *primitiva de petición* para el servicio A-ABORT ó una *primitiva de petición* para el servicio P-DATA. En consecuencia, la AE solicitante podrá recibir a través de su PSDUL: una *primitiva de confirmación* para el servicio A-RELEASE, una *primitiva de indicación* para el servicio A-ABORT ó una *primitiva de indicación* para el servicio P-DATA, respectivamente.

Si la *primitiva de respuesta* es para el servicio A-RELEASE, el parámetro de resultado contendrá el valor simbólico *afirmativo* (que indica la aceptación de la liberación).

Después de emitir la *respuesta* al servicio A-RELEASE, la AE solicitada no podrá enviar alguna otra primitiva dentro de la asociación, incluyendo la de *petición* al servicio P-DATA.

La AE solicitante recibirá una *primitiva de confirmación* al servicio A-RELEASE, a través de su PSDUL, con un valor *afirmativo* al parámetro de resultado.

Cualquiera de las dos AE's participantes en la asociación podrán interrumpir el servicio A-RELEASE, enviando una *primitiva de petición* al servicio A-ABORT. Cuando la AE receptora recibe una *indicación* al servicio A-ABORT, la asociación es liberada con la posible pérdida de la información en tránsito.

Se puede dar el caso de una colisión en el proceso del servicio A-RELEASE, cuando ambas AE's solicitan simultáneamente el servicio. Bajo estas circunstancias, ambas AE's recibiran una *primitiva de indicación* al servicio A-RELEASE en forma simultánea.

Para terminar la liberación de la asociación en forma normal, se debe realizar la siguiente secuencia:

- La AE solicitante en la asociación, emitirá una *primitiva de respuesta* al servicio A-RELEASE.
- La AE solicitada en la asociación, esperará una *primitiva de confirmación* para el servicio A-RELEASE de su contraparte. Cuando ésta llegue, emitirá una *primitiva de respuesta* para el servicio A-RELEASE.
- Por último la AE solicitante recibirá una *primitiva de confirmación* al servicio A-RELEASE.

La asociación se libera cuando ambas AE's hayan recibido una *primitiva de confirmación* al servicio A-RELEASE.

3.1.1.3. Servicio A-ABORT

Este es un *servicio no-confirmado* utilizado por una AE solicitante (culquiera de las dos participantes en la asociación) para forzar a una liberación anormal de la asociación.

Cuando un servicio A-ABORT es utilizado, la asociación debe liberarse en forma anormal, simultánea a la liberación anormal de la conexión de transporte.

Una AE que desee liberar la asociación en forma anormal emitirá una primitiva de petición para el servicio A-ABORT, cancelando la asociación y no permitiendo el envío de alguna otra primitiva. La AE receptora recibirá una primitiva de indicación para el servicio A-ABORT, a través de su PSDUL, incluyendo en el parámetro "Abort Source" (origen del aborto) el valor simbólico "*UL Service User*" que indica que fué iniciado por una AE solicitante. De esta forma, la asociación y la conexión de transporte también deben liberarse simultáneamente por parte de la AE receptora.

El PSDUL puede generar también una liberación anormal por errores internos. En tal caso, cualquiera de los PSDUL emitirá una primitiva de indicación hacia ambas AE's con un valor al parámetro "Abort Source" de "*UL Service Provider*". Este valor indica que el PSDUL inició la liberación anormal.

3.1.1.4. Servicio A-P-ABORT

Este servicio es utilizado únicamente por las interfases DUL para indicar una liberación anormal de la asociación. Esto puede ser a consecuencia de problemas

en los servicios de la capa de presentación (cierre inesperado de conexión) o en las capas inferiores. La ocurrencia de eventos de este tipo implica la posible pérdida de la información en tránsito.

Cuando un PSDUL detecta un error interno, éste emite primitivas de indicación para el servicio A-P-ABORT a ambas AE's y la asociación será liberada en forma anormal. De esta forma, no se podrá solicitar algún otro servicio.

3.1.1.5. Servicio P-DATA

Este es un servicio *no confirmado* y es utilizado por las AE's para realizar intercambio de información entre aplicaciones a través de mensajes DICOM. Después de haberse establecido la asociación entre dos AE's, cualquiera podrá emitir una primitiva de petición para el servicio P-DATA. La AE solicitada recibirá una primitiva de indicación para el mismo servicio. Para la transferencia de información se utiliza el parámetro: *lista de valores de datos de presentación*, que contiene uno o más PDV's ("Presentation Data Value") que a su vez contienen la identificación de un contexto de presentación y los valores de datos de usuario.

3.1.2. El protocolo DUL

El protocolo DUL incluye la funcionalidad de las capas de sesión, presentación y aplicación, del modelo ISO/OSI. De esta forma, DUL se define como protocolo de capa superior sobre TCP/IP y es una de las tres posibilidades de comunicación definidas en el estándar DICOM. El protocolo incluye en su especificación formatos de datos, una máquina de estados finitos y un temporizador para controlar tiempos de espera.

3.1.2.1. Formatos de datos

Los formatos de datos o PDU's ("Protocol Data Units") se utilizan para el intercambio de mensajes entre dos entidades dentro de una capa o protocolo. Los PDU's están especificados para contener información de control y datos definidos por la capa inmediata superior (Entidades de Aplicacion DICOM).

Los fotmatos de datos definidos para este protocolo son:

A-ASSOCIATE-RQ. Contiene la información necesaria para que una AE pueda solicitar una asociación a otra AE.

A-ASSOCIATE-AC. Contiene la información correspondiente a una respuesta de aceptación para una asociación entre dos AE's.

A-ASSOCIATE-RJ. Contiene la información correspondiente a una respuesta de rechazo a una petición de asociación.

P-DATA-TF. Contiene información relacionada a valores de datos de presentación, PDV's ("Presentation Data Value"), a transferir entre dos AE's.

A-RELEASE-RQ. Contiene información necesaria para que una AE pueda solicitar la liberación de la asociación a su contraparte.

A-RELEASE-RP. Contiene la información correspondiente a una respuesta de liberación de asociación entre dos AE's.

A-ABORT. Contiene la información correspondiente al aborto de una asociación entre dos AE's.

Los PDU's y la máquina de estados finitos ofrecen 3 servicios básicos a la capa de aplicación que son: establecimiento de asociación, transferencia de información y cierre de asociación.

En una asociación sólo pueden estar involucradas dos aplicaciones y en su establecimiento la aplicación solicitante emite, a través de DUL, un PDU A-ASSOCIATE-RQ para definir el tipo de asociación deseada (servicios solicitados). La aplicación receptora analiza el PDU recibido y responde con un PDU A-ASSOCIATE-AC, de aceptación, o con un PDU A-ASSOCIATE-RJ, de rechazo. La aplicación solicitante analiza la respuesta y, si es satisfactoria, se establece la asociación. De otra forma se rompe la comunicación.

En la transferencia de información, una vez establecida la asociación, las aplicaciones involucradas se comunican para intercambiar información. Las reglas de intercambio se definen durante el establecimiento de la asociación y se especifican como mensaje a transmitir a través del PDU P-DATA-TF. El contenido del mensaje es almacenado (completo o fragmentado) en un campo del PDU llamado PDV (Presentation Data Value) que es de longitud variable, debido a que el PDU puede incluir varios PDV's.

El cierre de una asociación puede realizarse desde cualquiera de las dos aplicaciones involucradas, utilizando 3 diferentes mecanismos: (1) La aplicación que inicia la asociación solicita su cierre emitiendo un PDU A-RELEASE-RQ, a través de DUL, esperando un PDU A-RELEASE-RP como confirmación; (2) Cualquier aplicación puede cerrar la asociación en forma anormal, a través de un PDU A-ABORT generado por DUL; (3) Cualquier aplicación puede terminar en forma anormal la asociación por alguna anomalia en la conexión, emitiendo un PDU A-ABORT a través de DUL.

Las especificaciones para los PDU's son:

- Deben construirse con campos mandatorios seguidos por campos opcionales de longitud variable, que contienen uno o más elementos y/o sub-elementos.
- Los elementos de tipo desconocido deben ignorarse y brincarse.
- Su codificación debe seguir las siguientes reglas:

    1. Cada PDU debe consistir de cadenas de bytes numerados secuencialmente a partir del número 1.
    2. Cada byte dentro del PDU debe consistir de 8 bits, numerados de 0 a 7, donde el bit 0 es el menos significativo.
    3. Cuando bytes consecutivos representan cadenas de caracteres, el primer caracter es representado por el número más pequeño correspondiente a su numeración.
    4. Cuando bytes consecutivos representan números binarios, el valor más significativo es representado por el número más pequeño correspondiente a su numeración (Formato "*Big Endian*", en orden de bytes).
    5. El byte representado por el número más pequeño, en la numeración del PDU, es colocado primero en el flujo de datos del servicio de transporte (TCP).

En las figuras 3.2 y 3.3 se observan los detalles de la estructura de cada PDU. De aquí puede considerarse la existencia de *PDU's compuestos* por elementos y sub-elementos (figura 3.2) y *PDU's simples* que contienen algún valor especificado (figura 3.3).

Los PDU's compuestos A-SSOCIATE-RQ y A-ASSOCIATE-AC (figura 3.2) contienen a los elementos *contexto de aplicación*, *contexto de presentación* e *información de usuario*. A su vez, el elemento contexto de presentación contiene a los sub-elementos *sintaxis abstracta* y *sintaxis de transferencia* y el elemento información de usuario al sub-elemento *máxima longitud*. De esta manera, también podemos considerar la existencia de *elementos simples* y *elementos compuestos* que se integran a los PDU's. Cada elemento y sub-elemento tiene un significado relevante para la capa de aplicación.

El PDU P-DATA-TF puede considerarse como compuesto porque contiene a elementos PDV, que a su vez se puede considerar como elemento simple con una estructura algo diferente a los demás elementos.

PDU's: A-ASSOCIATE-RQ y A-ASSOCIATE-AC

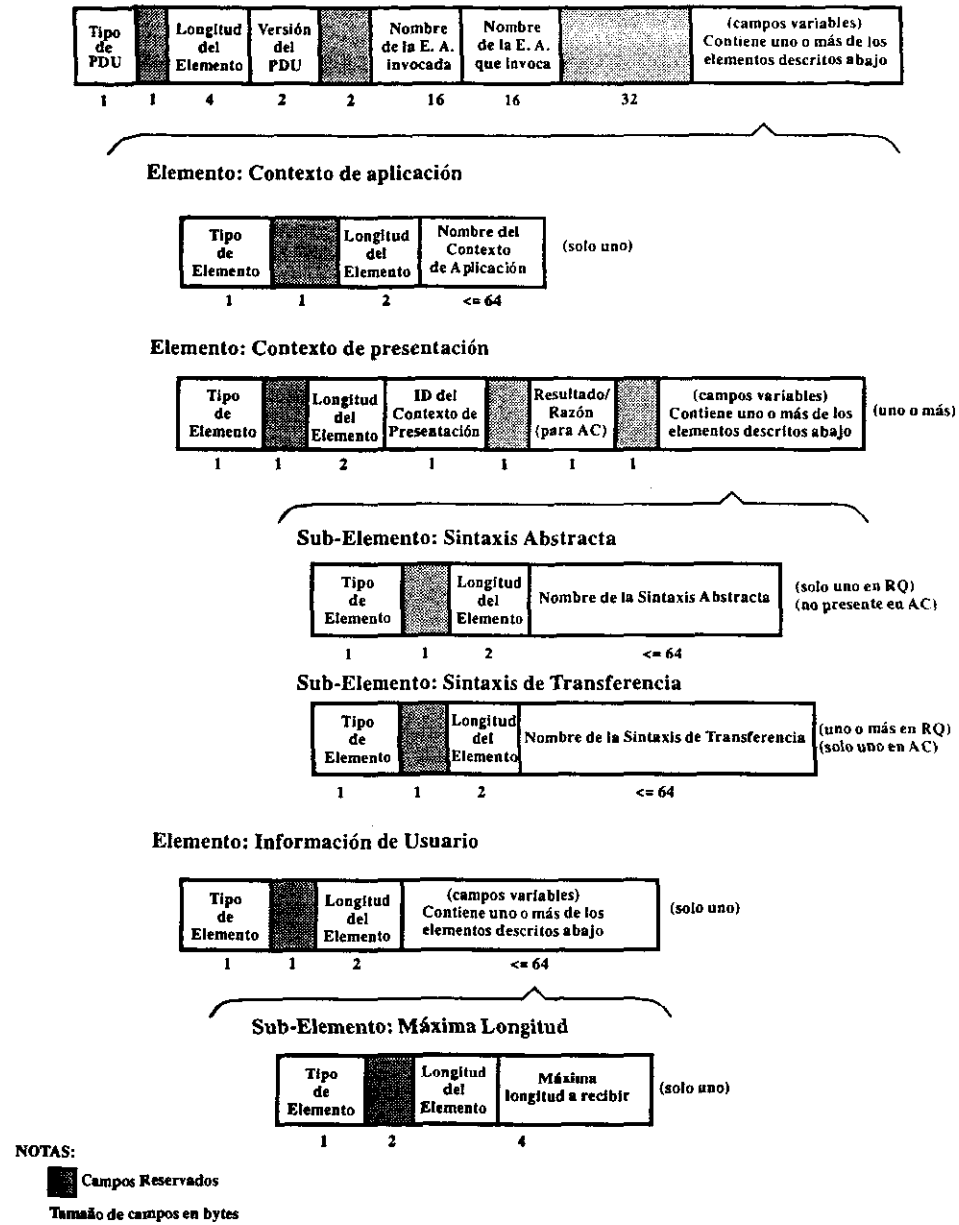
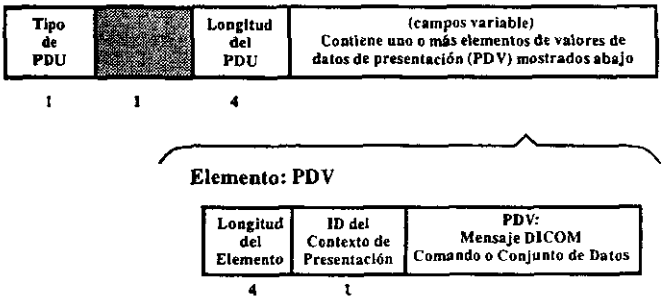


Figura 3.2: Estructura de los PDU's A-ASSOCIATE-RQ y A-ASSOCIATE-AC

PDU: P-DATA-TF



PDU's: A-ASSOCIATE-RJ / A-RELEASE-RQ / A-RELEASE-RP / A-ABORT

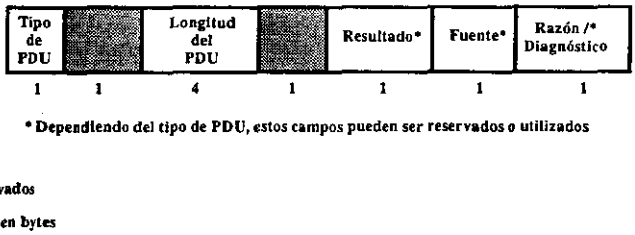


Figura 3.3: Estructura de los PDU's P-DATA-TF, A-ASSOCIATE-RJ, A-RELEASE-RQ, A-RELEASE-RP y A-ABORT

3.1.2.2. Máquina de estados finitos

La máquina de estados finitos del protocolo DUL esta especificada con 13 estados, puede recibir a 19 eventos y responder a 27 acciones diferentes. Se utiliza un temporizador especificado como *ARTIM* (*Association Request/Reject/Release Timer*) para medir tiempos de espera durante el proceso de asociación o liberación.

Estados Los estados están clasificados de la siguiente forma:

- **Estado ocioso**

  E–1 En espera de una *primitiva de petición al servicio A-ASSOCIATE* por parte de una AE ó de una *primitiva de indicación al servicio de apertura de conexión* por parte del transporte.

- **Estados para el establecimiento de asociación**

  E–2 Conexión de transporte ya establecida y se espera de un *PDU A-ASSOCIATE-RQ*.

  E–3 En espera de una *primitiva de respuesta A-ASSOCIATE* desde la AE asociada a DUL.

  E–4 En espera de una *primitiva de confirmación* para el servicio de apertura de conexión desde el transporte.

  E–5 En espera de un *PDU A-ASSOCIATE-AC ó A-ASSOCIATE-RJ*.

- **Estado para la transferencia de datos**

  E–6 Asociación ya establecida y se espera una *primitiva de petición al servicio P-DATA* desde la AE ó un *PDU P-DATA-TF* desde el transporte.

- **Estados para la liberación de la asociación**

  E–7 En espera de un *PDU A-RELEASE-RP* desde el transporte.

  E–8 En espera de una *primitiva de respuesta al servicio A-RELEASE* desde la AE asociada a DUL.

  E–9 En espera de una *primitiva de respuesta al servicio A-RELEASE* desde la AE asociada a DUL, después de resolver una colisión del lado del solicitante.

E–10 En espera de un *PDU A-RELEASE-RP* desde el transporte, después de resolver una colisión del lado del solicitado.

E–11 En espera de un *PDU A-RELEASE-RP* desde el transporte, después de resolver una colisión del lado del solicitante.

E–12 En espera de una *primitiva de respuesta al servicio A-RELEASE* desde la AE asociada a DUL, después de resolver una colisión del lado del solicitado.

- **Estado para la espera del cierre de conexión**

E–13 En espera de una *primitiva de indicación para el servicio de cierre de conexión* desde el transporte.

Eventos. De acuerdo a la función de DUL (figura 1.5), los eventos pueden llegar de la capa de transporte o de las AE's. También se considera un evento cuando termina el tiempo de espera medido por ARTIM.

- **Evento desde ARTIM**

Ev–1 Fin de tiempo de espera durante una asociación o liberación de asociación

- **Eventos desde el transporte**

Ev–2 Confirmación apertura de conexión.

Ev–3 Indicación de solicitud de conexión.

Ev–4 Indicación de cierre de conexión.

Ev–5 Recepción de un *PDU A-ASSOCIATE-RQ.*

Ev–6 Recepción de un *PDU A-ASSOCIATE-AC.*

Ev–7 Recepción de un *PDU A-ASSOCIATE-RJ.*

Ev–8 Recepción de un *PDU P-DATA-TF.*

Ev–9 Recepción de un *PDU A-RELEASE-RQ.*

Ev–10 Recepción de un *PDU A-RELEASE-RP.*

Ev–11 Recepción de un *PDU A-ABORT.*

Ev–12 Recepción de un PDU Desconocido..

- **Eventos desde las AE's**

  Ev–13 *Primitiva de petición al servicio A-ASSOCIATE.*

  Ev–14 *Primitiva de respuesta al servicio A-ASSOCIATE, aceptando el servicio.*

  Ev–15 *Primitiva de respuesta al servicio A-ASSOCIATE, rechazando el servicio.*

  Ev–16 *Primitiva de petición al servicio P-DATA.*

  Ev–17 *Primitiva de petición al servicio A-RELEASE.*

  Ev–18 *Primitiva de respuesta al servicio A-RELEASE.*

  Ev–19 *Primitiva de petición al servicio A-ABORT.*

Acciones. Las acciones para la máquina de estados finitos se clasifican en:

- **Acciones para el establecimiento de asociación**

  AE–1 Emite una *primitiva de petición de conexión* al transporte y se pasa al estado *E-4.*

  AE–2 Envía un *PDU A-ASSOCIATE-RQ* a través del transporte y se pasa al estado *E-5.*

  AE–3 Emite una *primitiva de confirmación al servicio A-ASSOCIATE* (de aceptación) a la AE solicitante y se pasa al estado *E-6.*

  AE–4 Emite una *primitiva de confirmación al servicio A-ASSOCIATE* (de rechazo) a la AE solicitante, cierra la *conexión de transporte* y pasa al estado *E-1.*

  AE–5 Emite una *primitiva de respuesta al servicio de conexión* al transporte, activa el *ARTIM* y se pasa al estado *E-2.*

  AE–6 Detiene el *ARTIM* y, si el *PDU A-ASSOCIATE-RQ* recibido del transporte es aceptable, emite una *primitiva de indicación al servicio A-ASSOCIATE* a la AE solicitada y se pasa al estado *E-3.* En otro caso, envía un *PDU A-ASSOCIATE-RJ* a través del transporte, activa el *ARTIM* y se pasa al estado *E-13.*

  AE–7 Envía un *PDU A-ASSOCIATE-AC* a través del transporte y se pasa al estado *E-6.*

AE–8 Envía un *PDU A-ASSOCIATE-RJ* a través del transporte y se pasa al estado *E-13.*

- **Acciones para la transferencia de datos**

DT–1 Envía un *PDU P-DATA-TF* a través del transporte y continua en el estado *E-6.*

DT–2 Emite una *primitiva de indicación al servicio P-DATA* a la AE solicitada y continua en el estado *E-6.*

- **Acciones para la liberación de la asociación**

AR–1 Envía un *PDU A-RELEASE-RQ* a través del transporte y pasa al estado *E-7.*

AR–2 Emite una *primitiva de indicación al servicio A-RELEASE* a la AE solicitada y pasa al estado *E-8.*

AR–3 Emite una *primitiva de confirmación al servicio A-RELEASE* a la AE solicitante, cierra la *conexión con el transporte* y pasa al estado *E-1.*

AR–4 Envía un *PDU A-RELEASE-RP* a través del transporte, activa el *ARTIM* y pasa al estado *E-13.*

AR–5 Detiene el *ARTIM* y se pasa al estado *E-1.*

AR–6 Emite una *primitiva de indicación al servicio P-DATA* a la AE solicitada y pasa al estado *E-7.*

AR–7 Envía un *PDU P-DATA-TF a través del transporte* y pasa al estado *E-8.*

AR–8 Emite una *primitiva de indicación al servicio A-RELEASE* a la AE solicitada (liberación de colisión), si el PDU A-RELEASE-RQ recibido es del solicitante, pasa al estado *E-9*; en otro caso pasa al estado *E-10.*

AR–9 Envía un *PDU A-RELEASE-RP* a través del transporte, y pasa al estado *E-11.*

AR–10 Emite una *primitiva de confirmación al servicio A-RELEASE* a la AE solicitante y pasa al estado *E-12.*

- **Acciones para abortar la asociación**

AA–1 Envía un *PDU A-ABORT* a través del transporte (como entidad solicitante), activa (o reactiva si ya se habia iniciado) *ARTIM* y pasa al estado *E-13.*

AA–2 Detiene el *ARTIM* si estaba activado, cierra la *conexión de transporte* y pasa al estado *E-1.*

AA–3 Si el solicitante de la asociación inicio el aborto, emite una *primitiva de indicación al servicio A-ABORT* a la AE solicitada y cierra la *conexión de transporte.* En otro caso (la AE solicitada inicio el aborto), emite una *primitiva de indicación al servicio A-P-ABORT* a la AE solicitante y cierra la *conexión de transporte.* En ambos casos pasa al estado *E-1.*

AA–4 Emite una *primitiva de indicación al servicio A-P-ABORT* a la AE y pasa al estado *E-1.*

AA–5 Detiene el *ARTIM* y pasa al estado *E-1.*

AA–6 Ignora el *PDU recibido* del transporte y pasa al estado *E-13.*

AA–7 Envía un *PDU A-ABORT* a través del transporte y pasa al estado *E-13.*

AA–8 Envía un *PDU A-ABORT* a través del transporte (como entidad solicitante), emite una *primitiva de indicación al servicio A-P-ABORT* a la AE asociada a DUL, activa el *ARTIM* y pasa al estado *E-13.*

En la figura 3.4, se observa un diagrama de estados reducido que resume la funcionalidad de la maquina de estados finitos para el protocolo DUL. Los círculos representan un estado o grupo de ellos y las transiciones indicadas con líneas y flechas, son etiquetadas con triadas Estado/Evento/Acción.

3.1.3. Interfase con TCP/IP

Para el uso de TCP/IP se especifica una relación uno a uno entre una conexión de transporte TCP y una asociación de capa superior, de acuerdo a las siguientes reglas:

- Cada asociación de capa superior (cliente) será soportada por una y solamente una conexión de transporte TCP.
- Cada conexión de transporte soportará una y solamente una asociación de capa superior (servidor).

El puerto TCP registrado por DICOM para DUL es el 104 (decimal). DICOM no sugiere el uso de una interfase específica para accesar los servicios de TCP.

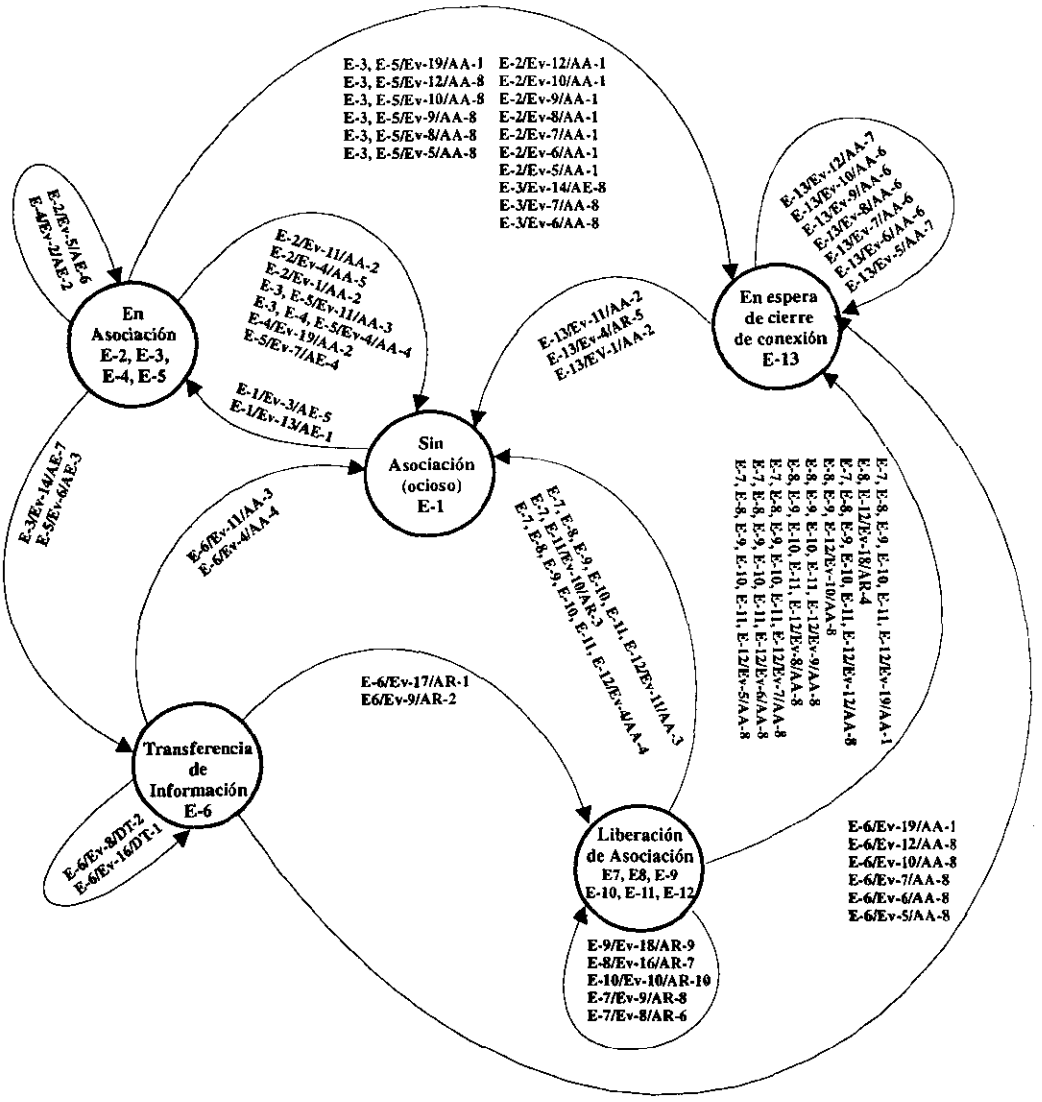


Figura 3.4: Diagrama de estados reducido

3.1.3.1. Apertura de una conexión de transporte TCP

Apertura Activa. Cuando una entidad de capa superior DICOM pretende establecer una asociación, debe emitir una primitiva de petición al servicio de conexión de transporte de TCP y esperar una primitiva de confirmación al mismo servicio. Una vez recibida la confirmación (conexión completada), se enviará información en la conexión de transporte establecida de acuerdo a las reglas establecidas para el protocolo DUL.

Apertura Pasiva. Cuando se activa una entidad de capa superior DICOM, ésta esperará una indicación al servicio de conexión al transporte en un modo pasivo. Cuando una indicación es recibida, ésta es aceptada y se inicializa el contador de tiempo ARTIM. Después se podrán intercambiar PDU's de acuerdo a las reglas establecidas para el protocolo DUL.

3.1.3.2. Cierre de una conexión de transporte TCP

Las conexiones TCP serán cerradas utilizando la opción *sin Demora* ("Don't Linger"). El cierre de una conexión TCP se puede dar por diferentes situaciones que son consideradas dentro de la máquina de estados finitos de DUL.

3.2. Análisis

Para propósitos del establecimiento de fronteras y como éste trabajo forma parte de un sistema con un nivel de complejidad mayor, se considera al *dominio del problema* como la parte que corresponde exclusivamente al protocolo DUL. A su vez, el protocolo considera dos interfases: una para enviar PDU's por red a través de TCP/IP y otra con las AE's DICOM que podrán hacer uso de sus servicios. De esta forma, la etapa de análisis se concentra en describir para el protocolo DUL: un contexto, un modelo del comportamiento deseado y un modelo de dominio.

3.2.1. Contexto para el protocolo DUL

La definición del contexto para el protocolo DUL se realizó utilizando diagramas de caso de uso y analizando sus diferentes instancias a través de diagramas de secuencia[1] ([Booch97], [Ambler97]).

[1]También llamados diagramas de trazo de mensajes, [Booch96].

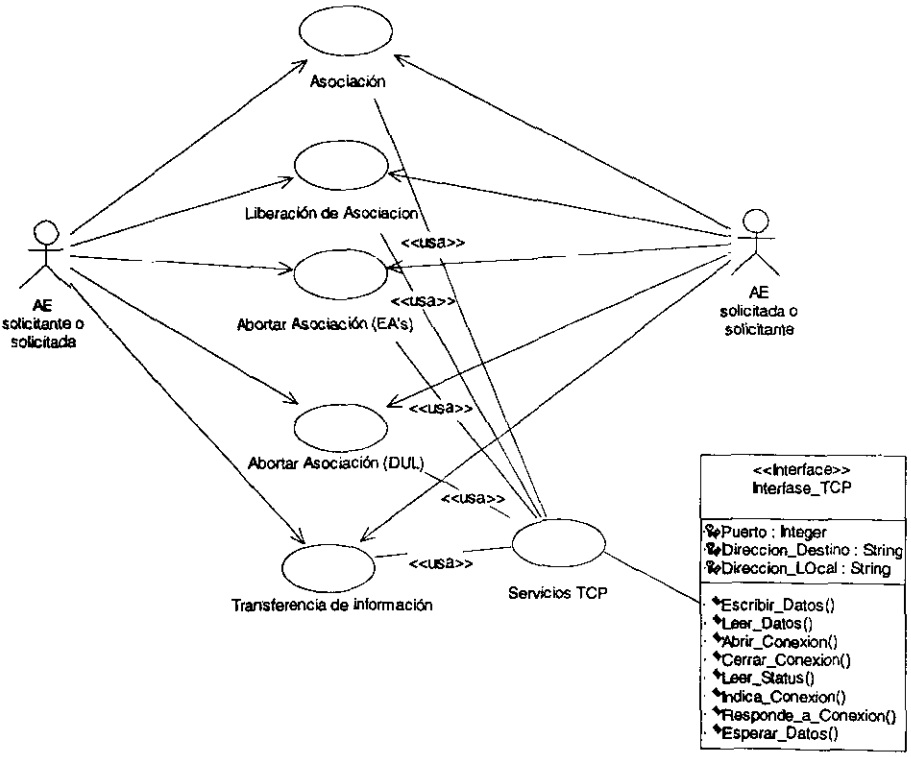


Figura 3.5: Diagrama de casos de uso

3.2.1.1. Casos de uso

Los casos de uso nos ayudan a entender la forma en la que las AE's interactuan con el protocolo DUL y cómo DUL hace uso de los servicios de TCP. La descripción de cada caso de uso se hizo de acuerdo a las especificaciones de la sección 3.1.1, para lo cual hacemos la siguiente correspondencia:

Asociación. Describe el caso de uso correspondiente al servicio A-ASSOCIATE.

Liberación de asociación Describe el caso de uso correspondiente al servicio A-RELEASE.

Aborto de la EA. Describe el caso de uso correspondiente al servicio A-ABORT.

Aborto del proveedor DUL. Describe el caso de uso correspondiente al servicio A-P-ABORT.

Transferencia de datos. Describe el caso de uso correspondiente al servicio P-DATA.

Servicios TCP. Se describe en la sección 3.1.3.

Los casos de uso se representan a través de diagramas que describen a tres elementos: actores, casos de uso y sus relaciones. Considerando a DUL como sistema en desarrollo, se contemplan 3 actores: AE solicitante, AE solicitada e Interfase TCP. Los casos de uso representan a todos los servicios ofrecidos por DUL a través de los servicios e interfase de TCP. En la figura 3.5, se representa el diagrama de casos de uso para el protocolo DUL, que incluye todos los elementos antes mencionados.

3.2.1.2. Instancias de casos de uso

Para entender con claridad la interacción del protocolo con el exterior, se plantean instancias de caso de uso a través de diagramas de secuencia. Los actores involucrados son dos AE's, solicitante y solicitada, además de dos proveedores de servicios del protocolo DUL asociados a cada AE. En nuestro caso, hacemos un mayor énfasis en las instancias de casos de uso en donde intervienen directamente las AE's que utilizan los servicios ofrecidos por DUL. Las instancias de caso de uso para los servicios de TCP se consideran implícitas en el envío de PDU's por parte de los proveedores del servicio DUL, porque son aspectos ya resueltos y únicamente se hace uso de ellos.

Las instancias de caso de uso estudiadas en este trabajo, de acuerdo a las especificaciones del protocolo, se describen a continuación utilizando diagramas

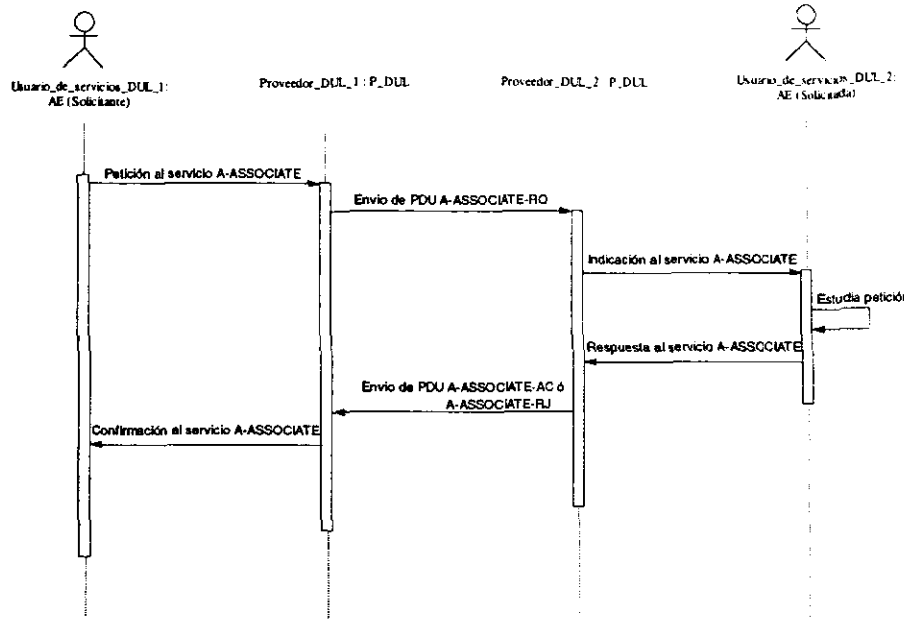


Figura 3.6: Asociación Normal

de secuencia. En los diagramas, los calificativos *solicitante* y *solicitado* asignados a las AE's, se refieren a la AE que solicita y a la AE que atiende una asociación, respectivamente.

Instancias del caso de uso asociación

- **Asociación normal.** En una asociación normal se invoca al servicio A-ASSOCIATE con las 4 primitivas y durante este proceso se establece una negociación de asociación entre dos AE's. La secuencia del proceso se observa en la figura 3.6.

- **Aborto de la AE solicitante antes de completar la asociación.** Puede darse el caso que la AE solicitante *se arrepienta* después de haber hecho una petición al servicio de asociación y haga uso del servicio de aborto para suspender dicha petición. En este caso la asociación no será establecida. Esta secuencia se observa en la figura 3.7.

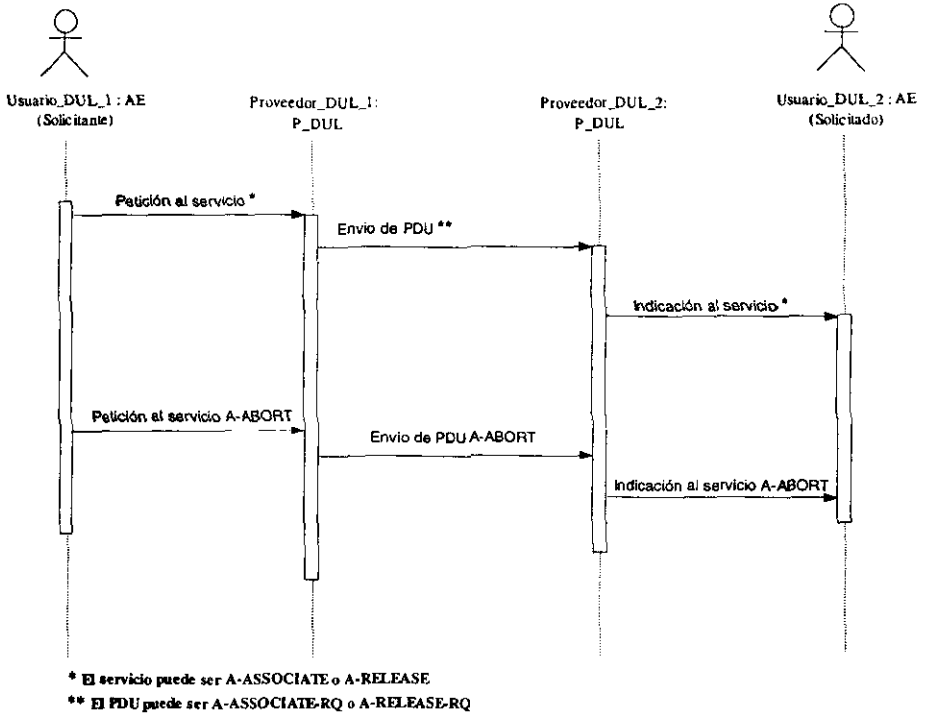


Figura 3.7: Aborto de la asociación o de la liberación de asociación por parte de la AE solicitante

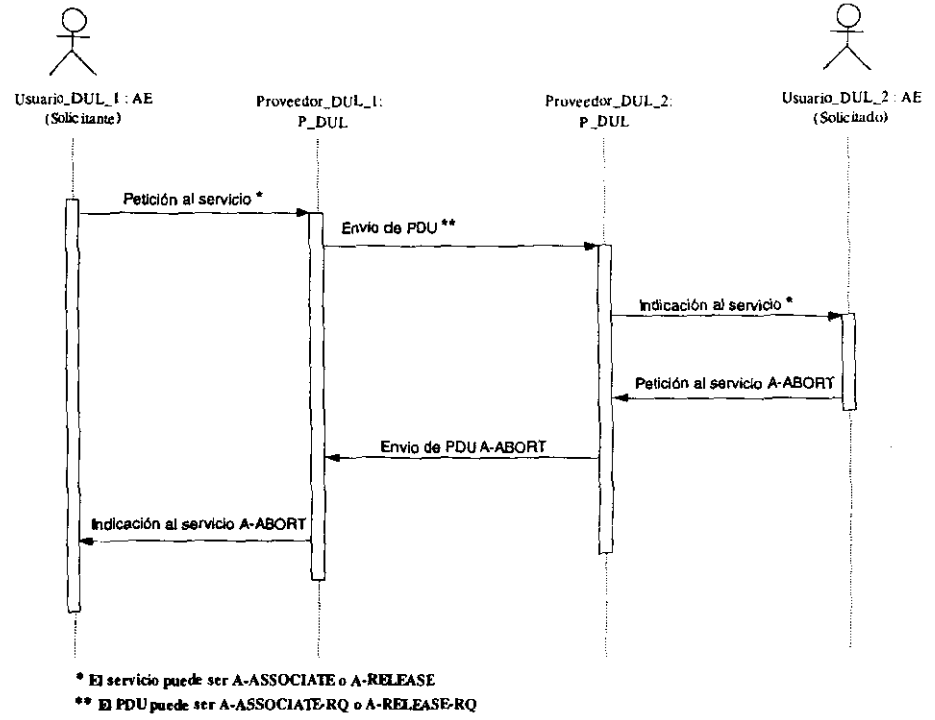


Figura 3.8: Aborto de la asociación o de la liberación de la asociación por parte de la AE solicitada.

- **Aborto de la AE solicitada antes de completar la asociación.** La AE solicitada podría abortar la asociación, por alguna razón anormal, después de recibir una primitiva de indicación al servicio A-ASSOCIATE. Esta secuencia se observa en la figura 3.8.

- **Asociación no soportada por el PSDUL.** En este caso, el proveedor de servicios DUL de la AE solicitada no es capaz de soportar la asociación y regresará una confirmación a la petición rechazando el servicio. Esta secuencia se observa en la figura 3.9.

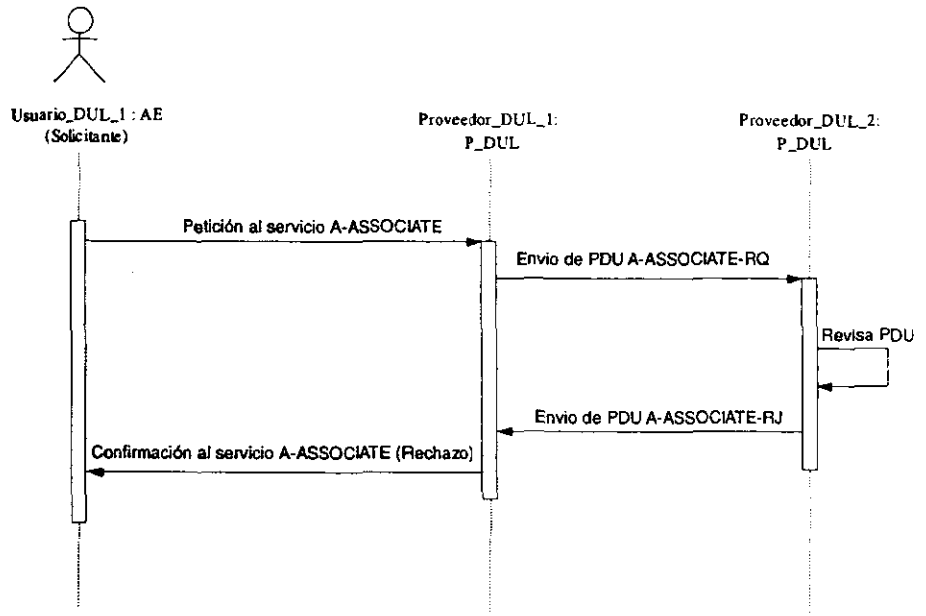


Figura 3.9: Asociación no soportada por el proveedor de servicios DUL

Instancias del caso de uso liberación de asociación

- **Liberación de asociación normal.** Este proceso se activa cuando cualquiera de las AE's participantes en la asociación desea liberarla. Para ello, la AE solicitante invoca al servicio A-RELEASE a través de una primitiva de petición y espera una confirmación. La AE solicitada recibe una indicación y después emite una respuesta. Esto se observa en la figura 3.10.

- **Aborto de la AE solicitante antes de completarse la liberación de asociación.** La AE solicitante del servicio de liberación de asociación puede interrumpir en forma rápida la asociación. Esta acción se realiza aún y cuando exista pérdida de la información en tránsito. Para ello, se emite una primitiva de petición al servicio A-ABORT que recibe la AE solicitada como una indicación al mismo servicio. Este mecanismo se ilustra en la figura 3.7

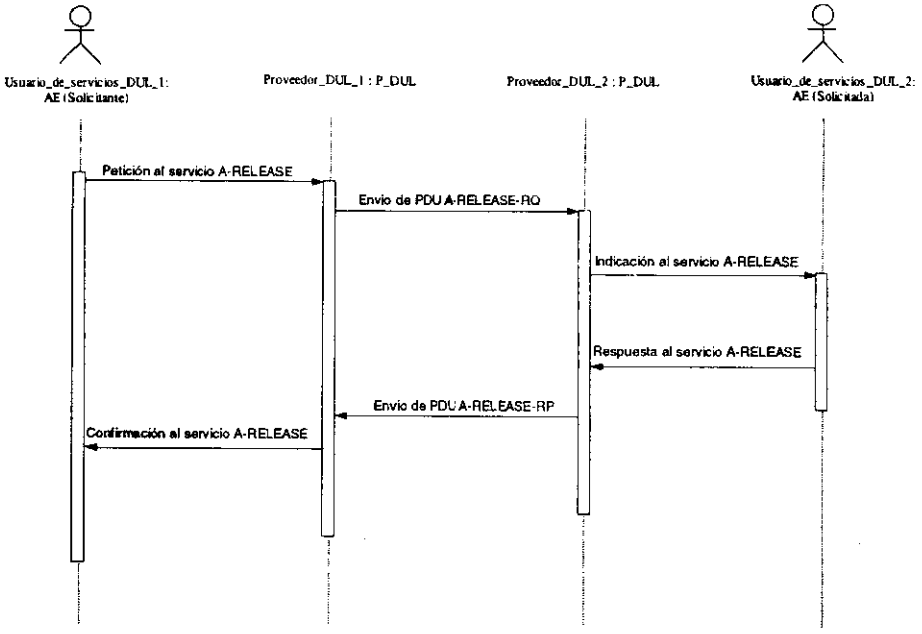


Figura 3.10: Liberación de asociación normal

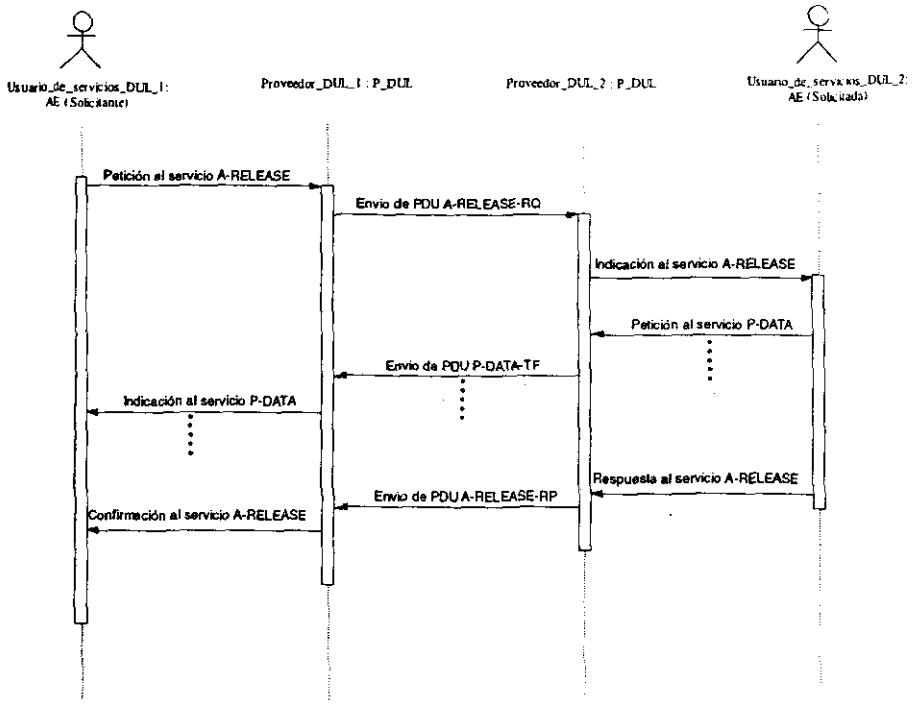


Figura 3.11: Transferencia de información antes de liberar la asociación

- **Aborto de la AE solicitada antes de completarse la liberación de asociación.** La AE solicitada emite una petición de aborto, después de recibir una indicación de liberación de asociación. Esto suspende la asociación en forma inmediata. Esta secuencia se observa en la figura 3.8.

- **Transferencia de información antes liberar la asociación.** En este caso, La AE solicitante hace una petición al servicio de liberación de asociación. La AE solicitada recibe la primitiva de indicación pero emite una primitiva de petición al servicio P-DATA, que es atendido por la AE solicitante. Cabe señalar que al realizarse el servicio P-DATA se invierten los papeles, convirtiendose la AE solicitante en solicitada y la AE solicitada en solicitante. Esto puede continuar hasta que la AE solicitada termina el servicio de liberación de asociación, emitiendo una primitiva de respuesta. Este mecanismo se observa en la figura 3.11.

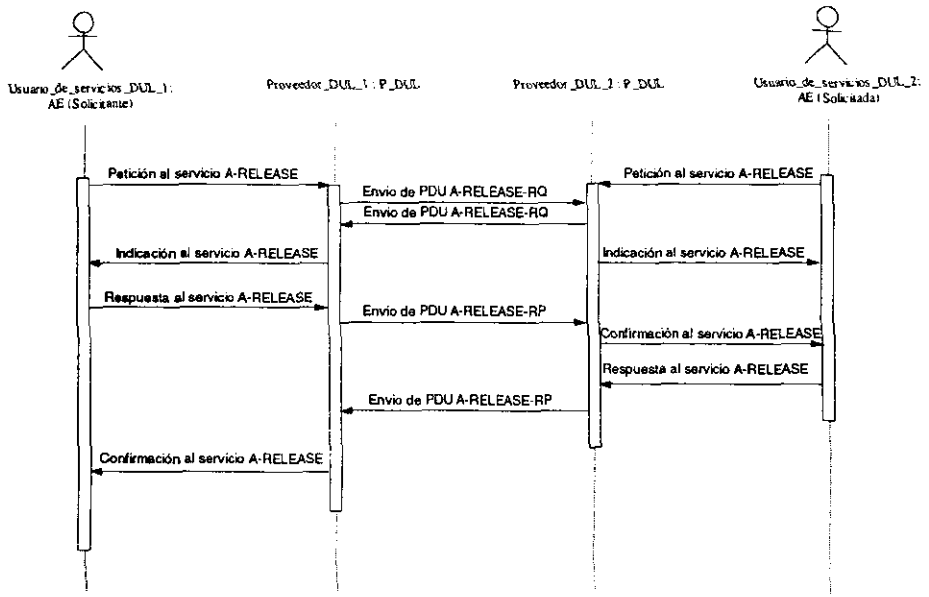


Figura 3.12: Colisión durante la liberación de asociación

- **Colisión en la liberación de asociación.** Una colisión se presenta cuando ambas AE's que participan en una asociación reciben al mismo tiempo una primitiva de indicación para liberar la asociación. En tal caso, la AE solicitante de la asociación emite primero su respuesta de liberación de asociación. Esta se recibe como una confirmación en la AE solicitada, quien emite inmediatamente su respuesta al mismo servicio. Este mecanismo se observa en la figura 3.12

Instancia del caso de uso aborto de la AE. Cualquier AE participante en la asociación puede emitir una primitiva de petición al servicio A-ABORT, que será recibido como una indicación del otro extremo. Esto se ilustra en las figuras 3.7 y 3.8.

Instancia del caso de uso transferencia de datos. Cualquier AE participante en la asociación puede emitir una primitiva de petición al servicio P-DATA, que será recibido como una indicación del otro extremo. Esta secuencia se incluye en la figura 3.11.

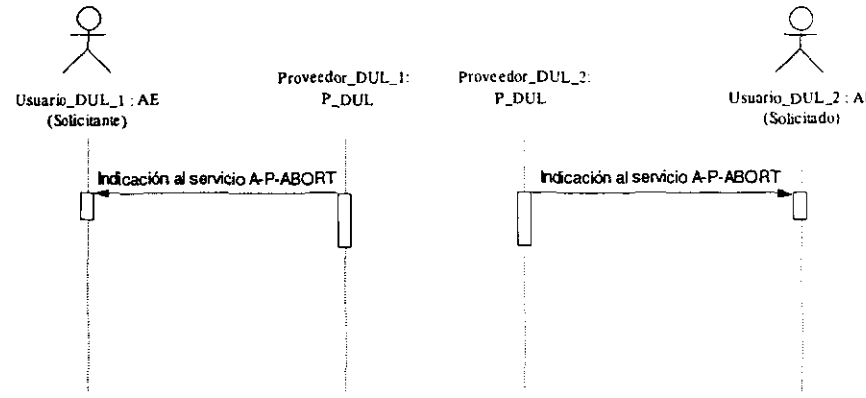


Figura 3.13: Aborto por parte del proveedor de servicios DUL

Instancia del caso de uso aborto del proveedor DUL. Cuando se detecta algún fallo en alguno de los dos proveedores de servicios DUL, se emite una indicación de aborto de asociación hacia la AE asociada al proveedor. El aborto se puede generar en cualquiera de los dos proveedores de servicios DUL involucrados. Este mecanismo se observa en la figura 3.13.

En los escenarios expresados a través de las figuras 3.6 a 3.13, se observa que cada uno de ellos no representa exclusivamente a una instancia de caso de uso, sino que, algunas veces representa a instancias de varios casos de uso.

La información proporcionada por estos escenarios hace evidente la forma en que se utilizan los servicios del protocolo DUL por AE's y como DUL utiliza los servicios de TCP, representando así el contexto del protocolo DUL.

3.2.2. Modelo de comportamiento DUL

Después de entender con claridad la interacción del protocolo hacia el exterior a través de los escenarios planteados anteriormente, se intentó encontrar, en primera instancia, el *comportamiento hacia el interior utilizando tarjetas CRC* como herramienta. Este proceso sirvió para empezar a descubrir abstracciones dentro del contexto del protocolo, asignándoles comportamiento y encontrando la colaboración entre ellas.

El comportamiento a más alto nivel de abstracción se propone para el protocolo como un todo. De esta forma, DUL es capaz de recibir y enviar información correspondiente a los servicios a través de primitivas y, a su vez, requiere enviar

y recibir información a través del transporte TCP.

Hacia el interior del protocolo se observa: el vencimiento de tiempo fuera y la llegada de información de TCP y de las AE's. Este comportamiento genera eventos que debe recibir una *máquina de estados finitos* (MEF). De acuerdo al evento recibido, la MEF emitirá acciones que se traducen en envío de información hacia TCP y AE's, asi como la manipulación de ARTIM. De acuerdo a la recepción de un evento válido y al estado actual, la MEF define el siguiente estado y la acción a ejecutar.

El comportamiento básico de ARTIM es iniciar, arrancar, detener y reiniciar. Por otra parte, en los formatos de datos se especifica el cálculo de longitud de los PDU's y la conversión a "Big Endian" de valores binarios que forman parte del formato. En consecuencia, un PDU debe garantizar una conformación válida antes de ser anviado hacia la AE local o remota.

De este análisis se hacen evidentes siete abstracciones: interfase TCP, protocolo DUL, ARTIM, MEF, AE, mapa de estados y formato de datos (PDU). Los detalles de asignación de responsabilidades y de colaboración entre ellas, se describen en las tarjetas CRC siguientes:

clase: Interfase TCP	
Responsabilidades	*Colaboradores*
Manejo de Conexión	TCP
Envío de datos	TCP
Recepción de datos	TCP
Manejo de Estatus	TCP

clase: Protocolo DUL	
Responsabilidades	*Colaboradores*
Recepción de información de TCP	Interfase TCP
Recepción de información de AE's	AE
Manejo de Servicios	MEF

clase: ARTIM	
Responsabilidades	*Colaboradores*
Arranque	S.O.
Paro	ARTIM
Reinicio	S. O.

clase: MEF	
Responsabilidades	*Colaboradores*
Cambio estado	Mapa
Recepción de eventos	Protocolo DUL
Ejecución de acciones	Interfase TCP, AE

clase: AE	
Responsabilidades	*Colaboradores*
Solicitud de servicios DUL	Protocolo DUL
Recepción de indicaciones y confirmaciones de servicio	Protocolo DUL

clase: Mapa	
Responsabilidades	*Colaboradores*
Determinación del siguiente estado	Mapa
Determinación de la siguiente acción	Mapa

clase: Formato	
Responsabilidades	*Colaboradores*
Conversión a "big Endian"	Formato
Cálculo de longitud	Formato
Validación de PDU	Formato

Para afinar la asignación de comportamiento, se tomó la decisión de comprobar el servicio confirmado A-ASSOCIATE normal. La prueba se realizó en dos partes: la primera corresponde a la solicitud de la asociación por parte de una AE y la segunda corresponde al procesamineto de dicha petición por parte del PSDUL de la AE solicitada.

3.2.2.1. Comprobación de una solicitud de asociación del lado del solicitante

En el diagrama de secuencia de la figura 3.14, se observa cómo una *AE solicitante*, antes de realizar alguna petición de servicio, contruye un objeto *Proveedor_DUL_1* de tipo *P_DUL* y este a su vez a sus componentes: *TM1* de tipo *ARTIM*, *M1* de tipo *MEF* y *TCP1* de tipo *Interfase_TCP*. *M1* construye a *MP1* de tipo *Mapa*, que representa su mapa de estados y dá a conocer a *M1* *TM1* y *TCP1*, colocando finalmente a *M1* en un estado *ocioso* inicial (ver 3.1.2.2).

Posteriormente, la *AE solicitante* realiza una petición al *servicio A-ASSOCIATE* a través del objeto *Proveedor_DUL_1* quien lo traduce en evento para *M1*. *M1* consulta a *MP1* para determinar cuál será su estado siguiente y la acción a realizar de acuerdo al estado actual y al evento recibido. En este caso, *MP1* le hace saber a *M1* que la acción a realizar es: solicitar a *TCP1* la *apertura de conexión* y regresar una *indicación de apertura de conexión*.

Una vez que la apertura de conexión se ha realizado, el *Proveedor_DUL_1*, utilizando la información que recibió en la petición del servicio, produce un *evento para M1* que consiste en la petición al *servicio A-ASSOCIATE*. *M1* consulta a *MP1* para determinar su siguiente estado y la acción a ejecutar. En este caso, la acción consiste en mandar un *PDU A-ASSOCIATE-RQ* y apoyándose en el objeto *PDU1* de tipo *Formato*, para crearlo (con los parámetros recibidos en la petición al *servicio A-ASSOCIATE*), validarlo y serializarlo, es enviado a través de *TCP1*.

Después de enviar el PDU, el *Proveedor_DUL_1* permanece verificando la llegada de datos a través de *TCP1* que se traduzcan en la recepción de un PDU de *confirmación al servicio A-ASSOCIATE*. Cuando los datos llegan, se genera un *evento para M1*. *M1* vuelve a consultar a su mapa de estados *MP1*, cambia de estado y la acción a ejecutar es: crear un PDU a partir de los datos de confirmación recibidos, validarlo, serializarlo y enviarlo, a la AE solicitante.

3.2.2.2. Comprobación de una solicitud de asociación del lado del solicitado

En la figura 3.15, se observa un *proceso concurrente* que se encarga de *escuchar* peticiones de conexión de procesos remotos. Cuando se tiene una nueva *petición de conexión*, el proceso concurrente inicia a la *AE solicitada* que atenderá las peticiones respectivas.

La *AE* genera a un objeto *Proveedor_DUL_2* de tipo *P_DUL* que desencadena la construcción de los objetos restantes que colaboran con el protocolo: *TM2* de tipo *ARTIM*, *M2* de tipo *MEF*, con su mapa de estados respectivo *MP2*, y *TCP2* de tipo *Interfase_TCP*. Una vez construidos los objetos, se asigna a *M2* un estado inicial (ocioso) y se le da a conocer a *TM2* y *TCP2*.

Posteriormente, el *proceso concurrente* le pasa el control a la *AE solicitada* para atender las peticiones de servicio a través del *Proveedor_DUL_2* quien, ya con el control, le hace saber a *M2* el evento *indicación de conexión*. *M2* consulta a *MP2* para determinar su estado siguiente y la acción a ejecutar. En este caso, *M2* responde a la *indicación de conexión* a través de *TCP2*, arranca al temporizador *TM2* y le regresa el control al *Proveedor_DUL_2*, quien permanece verificando la

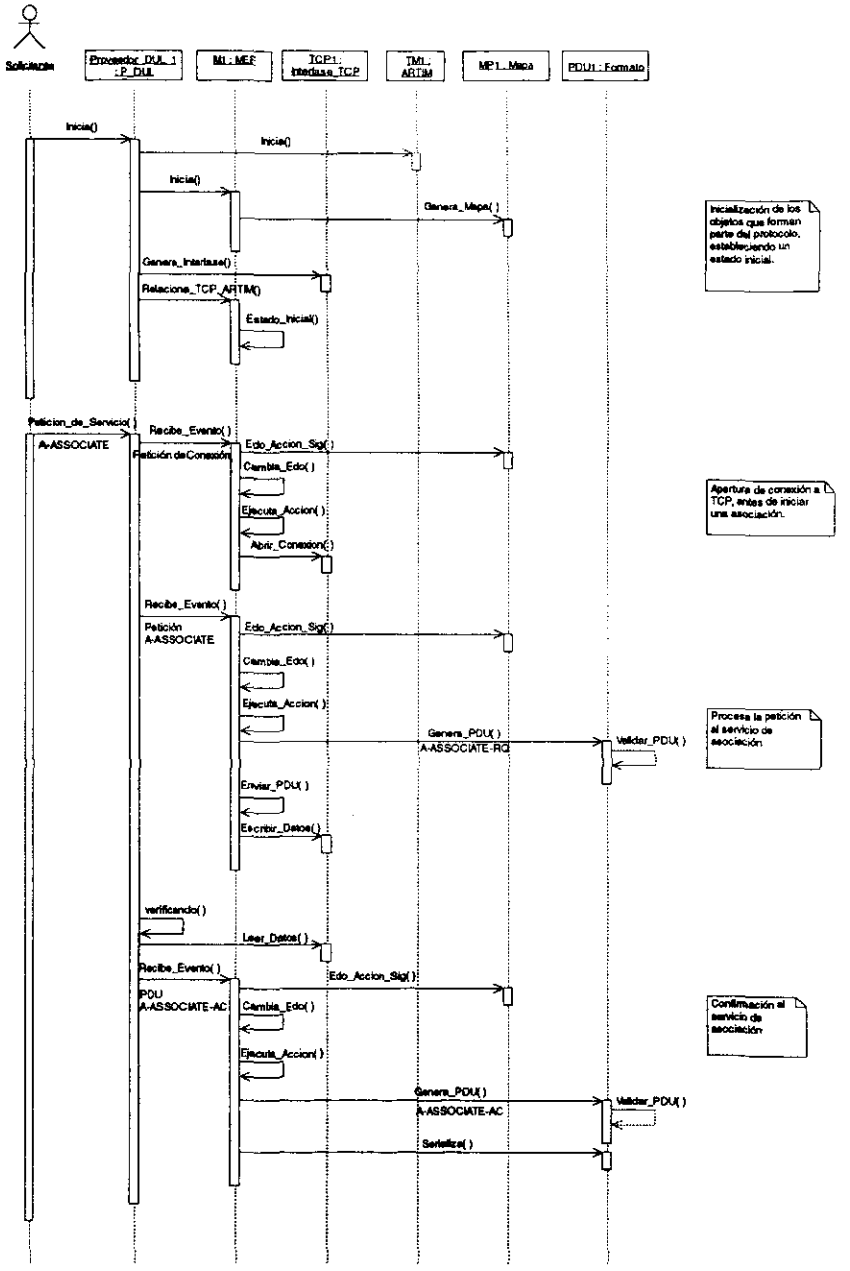


Figura 3.14: Petición de Asociación

llegada de un PDU desde *TCP2* y el *tiempo fuera* a través de *TM2*.

Cuando el objeto *Proveedor_DUL_2* detecta la llegada de un PDU, lo envía como evento a *M2* quien vuelve a consultar a *MP2* para determinar el estado siguiente y la acción a ejecutar. En este caso consiste en: parar a *TM2*, generar un *PDU A-ASSOCIATE-RQ*, validarlo, serializarlo y enviarlo a la AE solicitada.

La *AE* manda su respuesta al objeto *Proveedor_DUL_2*, quien lo traduce en un evento para *M2* que cambia al siguiente estado. La acción a ejecutar en este caso consiste en: generar, validar, serializar y enviar un *PDU A-ASSOCIATE-AC* para completar la asociación.

3.2.2.3. Observaciones

De este análisis se desprenden las siguientes observaciones:

- Cada vez que la Máquina de Estados Finitos recibe un evento, éste debe recibir la información suficiente para identificarlo. De esta forma, el cambio al siguiente estado y la ejecución de cada acción siempre se realizan bajo el mismo patrón (ver figuras 3.14 y .3.15).

- La MEF debe guardar información relacionada a su estado actual, para que, a través de su mapa de estados pueda decidir cuál es el siguiente estado y la acción a ejecutar.

- Se considera a los objetos que representan al protocolo DUL (*Proveedor_DUL_1 y Proveedor_DUL_2*) como los responsables de ofrecer los servicios a AE's. En la petición de un servicio, la identificación del éste se realiza a través de parámetros que recibe el protocolo y cuando regresa el control al objeto *solicitante* de tipo AE, éste recibe información correspondiente a una confirmación si el servicio es confirmado.

- Se considera que todos los eventos llegan primero a los objetos que representan el protocolo DUL (*Proveedor_DUL_1 y Proveedor_DUL_2*), quien los da a conocer a su objeto *MEF*. Sin embargo, las acciones las ejecuta directamente el objeto *MEF*, por lo que ésta debe tener una liga con los objetos que representan al temporizador (*ARTIM*) y a la interfase *TCP*.

- Los objetos que representan al formato de datos o *PDU's*, sólo son creados si son requeridos.

- En el diagrama de la figura 3.15 se estableció la existencia de un *proceso concurrente* del cual no se ofrecen detalles. Este proceso verifica si algún *proceso local* o *remoto* desea establecer una conexión a través de TCP.

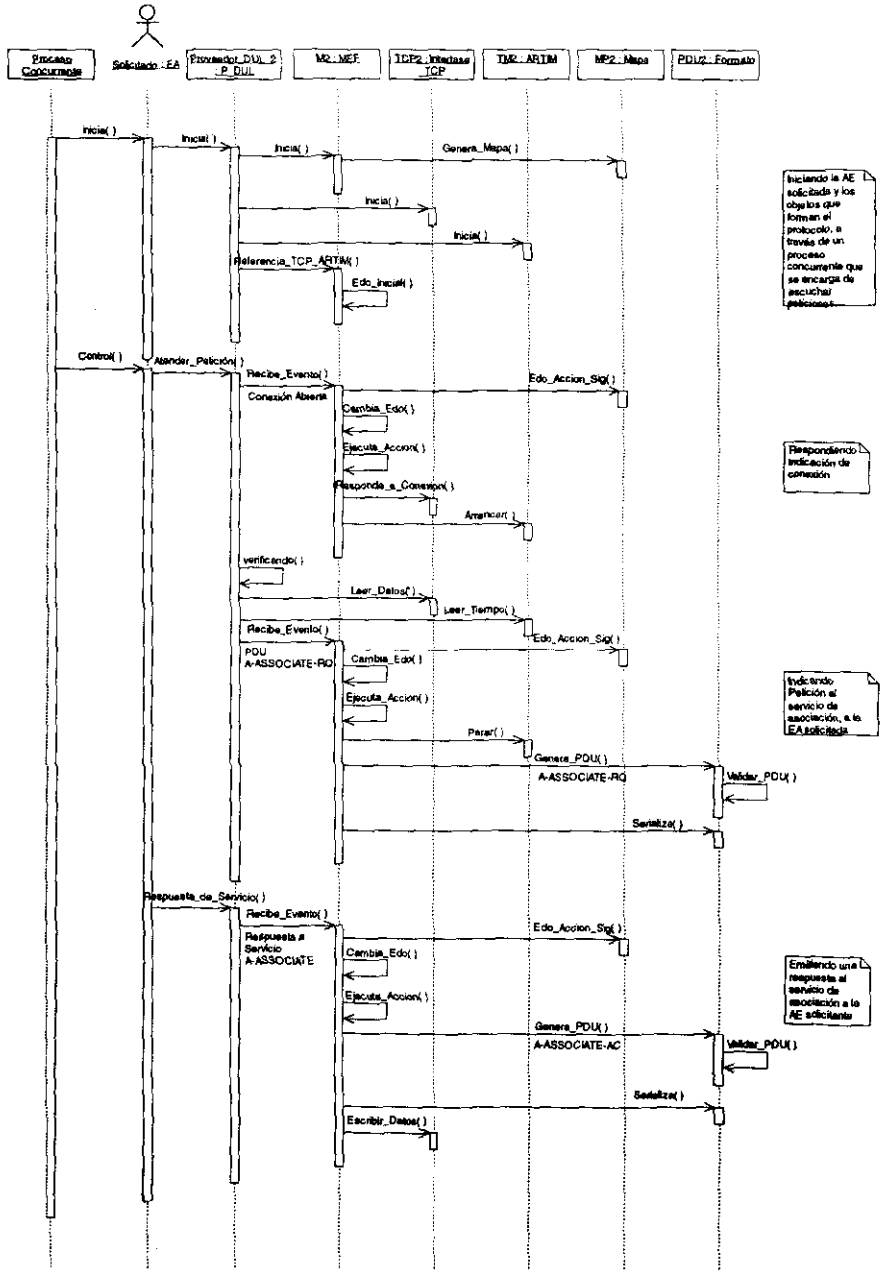


Figura 3.15: Proceso de atención de asociación por parte del solicitado.

Considerando que por cada asociación debe existir una conexión (ver sección 3.1.3), este proceso *creará una AE* que atienda una asociación por cada *petición de conexión.*

- De acuerdo a la observación anterior, una vez que el *proceso concurrente* le pasa el control a la AE que atenderá la asociación, ésta considera la recepción de un evento que consiste en una indicación de conexión, entregándola al objeto que representa al protocolo DUL.

- Las primitivas de *petición, indicación, respuesta* y *confirmación* para los servicios confirmados, son *implícitas* en cada responsabilidad asignada a los objetos que representan al protocolo DUL.

3.2.3. Modelo de dominio DUL

De acuerdo a los resultados obtenidos anteriormente, se propone el diagrama de clases de la figura 3.16 en donde se observan las responsabilidades y roles asignados a cada una de ellas (obtenidos principalmente de las tarjetas CRC).

Las relaciones entre clases que se plantean en el diagrama son:

- La clase *P_DUL* tiene agregadas por valor a las clases *MEF* y *ARTIM.*

- La clase *P_DUL* mantiene asociaciones con la clase *AE* y la clase *interfase_TCP.*

- La clase *MEF* establece relaciones de uso (o dependencia) con *ARTIM* e *Interfase_TCP*, para manejar las acciones.

- La clase *MEF* tiene agregada por valor a la clase *Mapa*, que representa a su mapa de estados.

- La clase *MEF* establece una relación de uso con la clase *Formato*, para crear PDU's cuando la acción a ejecutar lo requiera.

Con respecto a los Formatos de datos se descubrieron el comportamiento y estructura siguientes:

- Se consideran 6 clases: *Formato*, *Elemento_Simple*, *Elemento_Compuesto*, *Sub-Elemento*, *PDU_Simple* y *PDU_Compuesto.*

- Las relaciones entre clases muestran una gran similitud entre todos los formatos especificados para DUL, lo que define una jerarquía de clases a través

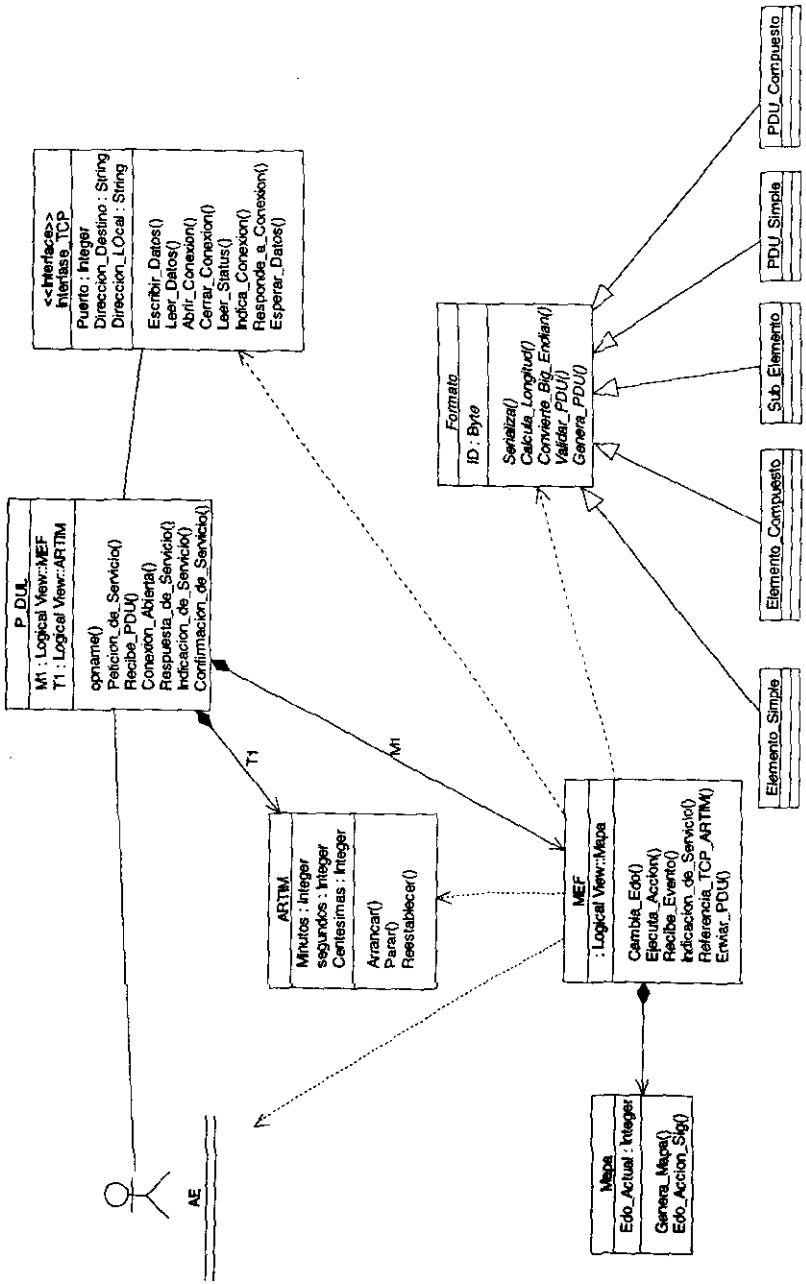


Figura 3.16: Modelo de dominio

del mecanismo de herencia. La clase de más alta jerarquía, *Formato*, representa los aspectos comunes de todos los formatos. Estos incluyen: tipo, longitud y campos reservados. Esta estructura se observa en la figura 3.16.

- Las subclases definen básicamente a todos los tipos de PDU's, elementos y sub-elementos. Además, cada subclase mantiene sus atributos particulares, responsabilizandose en darles un valor apropiado.

- Las responsabilidades más importantes de todos los objetos *Formato* son calcular su longitud y garantizar el orden de bytes apropiado para su transmisión en red. Todos los objetos de tipo *PDU_Simple* y *PDU_Compuesto* deben, de alguna manera, llenar sus atributos para posteriormente calcular su longitud. La cuenta de longitud comienza a partir del siguiente campo con respecto al campo de longitud. Como se observa en las figuras 3.2 y 3.3, la dificultad para calcular la longitud es mayor para los PDU's compuestos, debido a que contienen elementos simples, compuestos y sub-elementos. Sin embargo, al delegar a cada objeto el cálculo de su longitud, el problema se simplifica.

- Por otro lado, cada objeto de tipo *Formato*, debe garantizar el envío de sus datos binarios de más de 2 bytes en un orden de bytes "*Big Endian*". La solución a este problema también se simplifica si se responsabiliza a cada objeto de su realización, principalmente para los objetos compuestos.

3.3. Diseño

3.3.1. Diseño de los formatos de datos DICOM

En las figuras 3.2 y 3.3 se observan las estructuras de los formatos de datos o PDU's, especificadas para el protocolo DUL. La figura 3.2 muestra que los formatos más complejos son los involucrados en el servicio de asociación. En las especificaciones de codificación planteadas en la seccion 3.1.2.1 se establece que sus valores binarios deben codificarse en formato de orden de bytes "*Big Endian*" y que debe calcularse la longitud de cada elemento que forma parte de los PDU's.

Desde la perspectiva del protocolo DUL, para realizar cada servicio ofrecido, requiere de algún PDU. Sin embargo, no se puede determinar apriori el servicio que se solicitará. En consecuencia, se requiere que con solo conocer el servicio solicitado se construya un PDU apropiado en forma automática. Además la manipulación de los PDU's, en cuanto a la recepción y envío, se debe realizar en forma serializada a través de la interfase con TCP.

Para cumplir con estos requerimientos, estructurales y de comportamiento, de [Gamma95] se seleccionaron básicamente tres patrones de diseño que se describen a continuación.

3.3.1.1. Patrón "Composite"

Es un patrón que genera objetos compuestos con base a una estructura de árbol para representar jerarquías de *todo/partes*. Esto permite a objetos simples y compuestos ser manejados uniformemente. Para este trabajo se han definido PDU's y elementos, simples y compuestos, para los que se propone la estructura mostrada en la figura 3.18. En esta estructura podemos considerar *elementos primitivos* a las clases: *PDU_Simple*, *Elemento_ Simple* y *PDV*.

En la clase *formato*, considerada *clase abstracta*, se incluyen todos los atributos comunes a sus derivados, como son: tipo, un campo reservado y longitud. En sí, declara la interfase para todos los objetos de la composición.

Como puede observarse a partir de las figuras 3.2 y 3.3, los elementos primitivos, además de los atributos heredados, contienen la información proveniente de la AE, aunque el elemento *PDV* no utiliza los campos de longitud.

La clase *Formato_Compuesto*, estereotipada como *<<composite>>*, es la encargada de definir a los elementos compuestos a través de un *tipo de dato abstracto* (TDA) *Cola*. Las clases compuestas (*PDU_ASSOC*, *PDU_P_DATA* e *Info_Usr*) contienen en una cola todos sus componentes simples o compuestos, lo cual genera una estructura compleja y arbitraria en algunos casos (objetos con una cola de elementos, que a su vez contienen una cola con elementos). En estas clases se define el comportamiento de elementos y PDU's con componentes e implantan las operaciones virtuales relacionadas definidas en la interfase.

3.3.1.2. Patrón "Factory Method"

Este patrón ofrece una interfase para crear un objeto, pero deja a las subclases decidir cual clase instanciar, delegando así la creación a clases derivadas.

Cada vez que un objeto de *P_DUL* requiere de un PDU envía un mensage *Formato::Genera_PDU()* para construir un formato de algún tipo especifico, considerando que el método *Genera_PDU()* es de clase. Sin embargo, esta petición se produce en tiempo de ejecución y este patrón, expresado en la clase abstracta *Formato*, se encarga de construir el PDU apropiado de acuerdo a la información recibida por el servicio solicitado en un objeto de tipo *buffer*, ofreciendo flexibilidad en esta actividad.

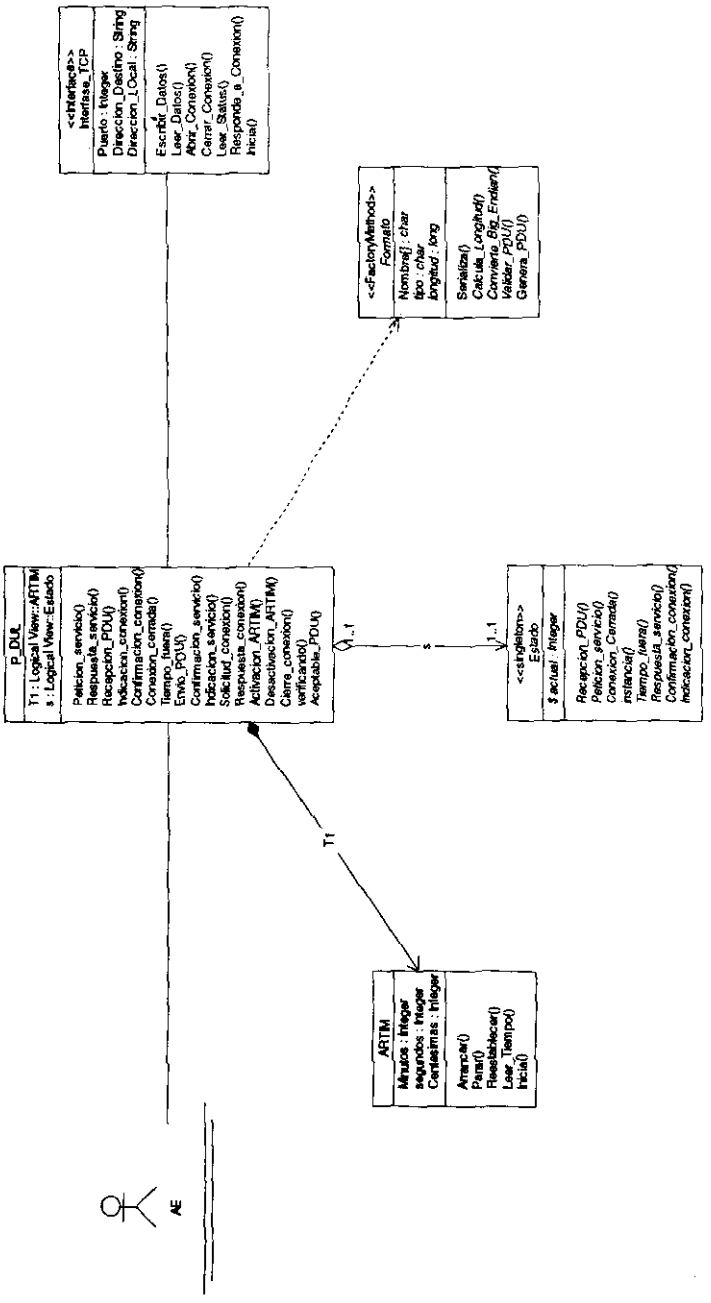


Figura 3.17: Clases básicas que definen el protocolo

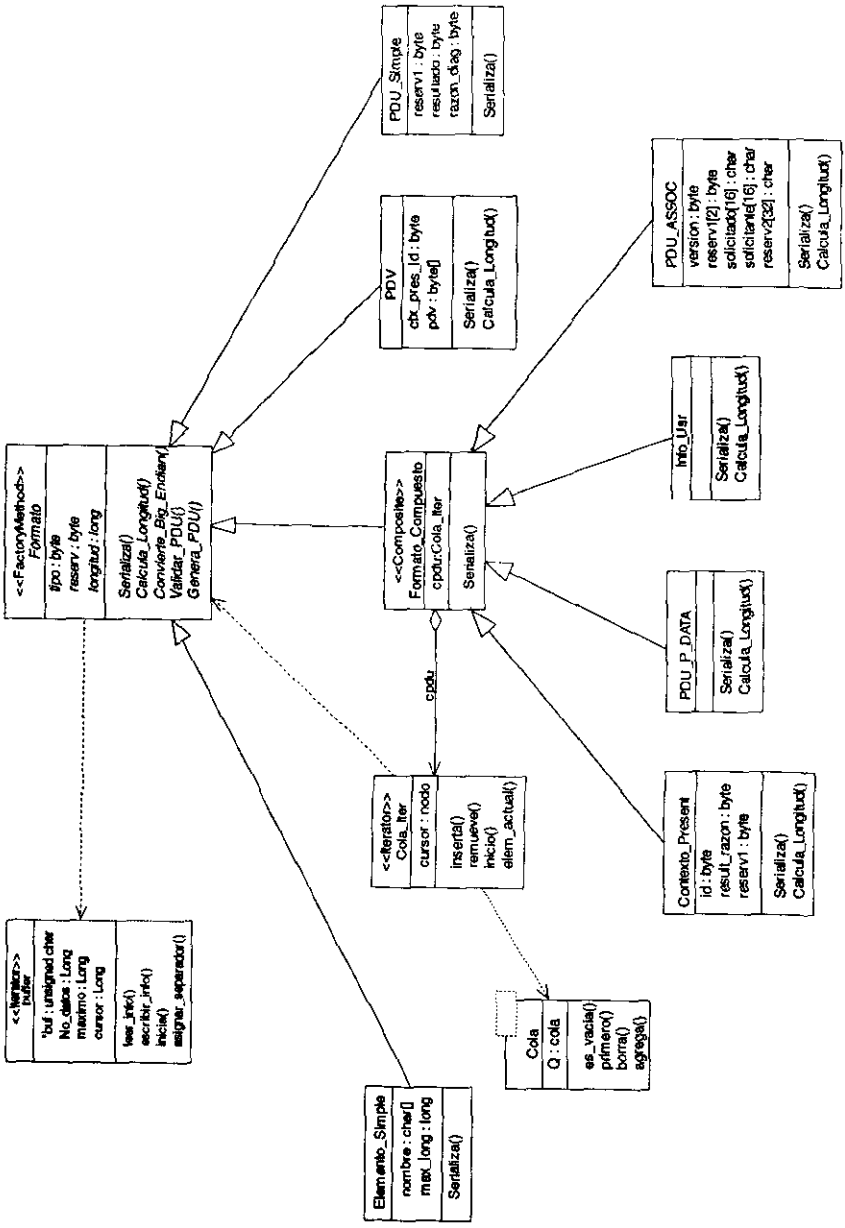


Figura 3.18: Estructura de clases para los formatos de datos (PDU's).

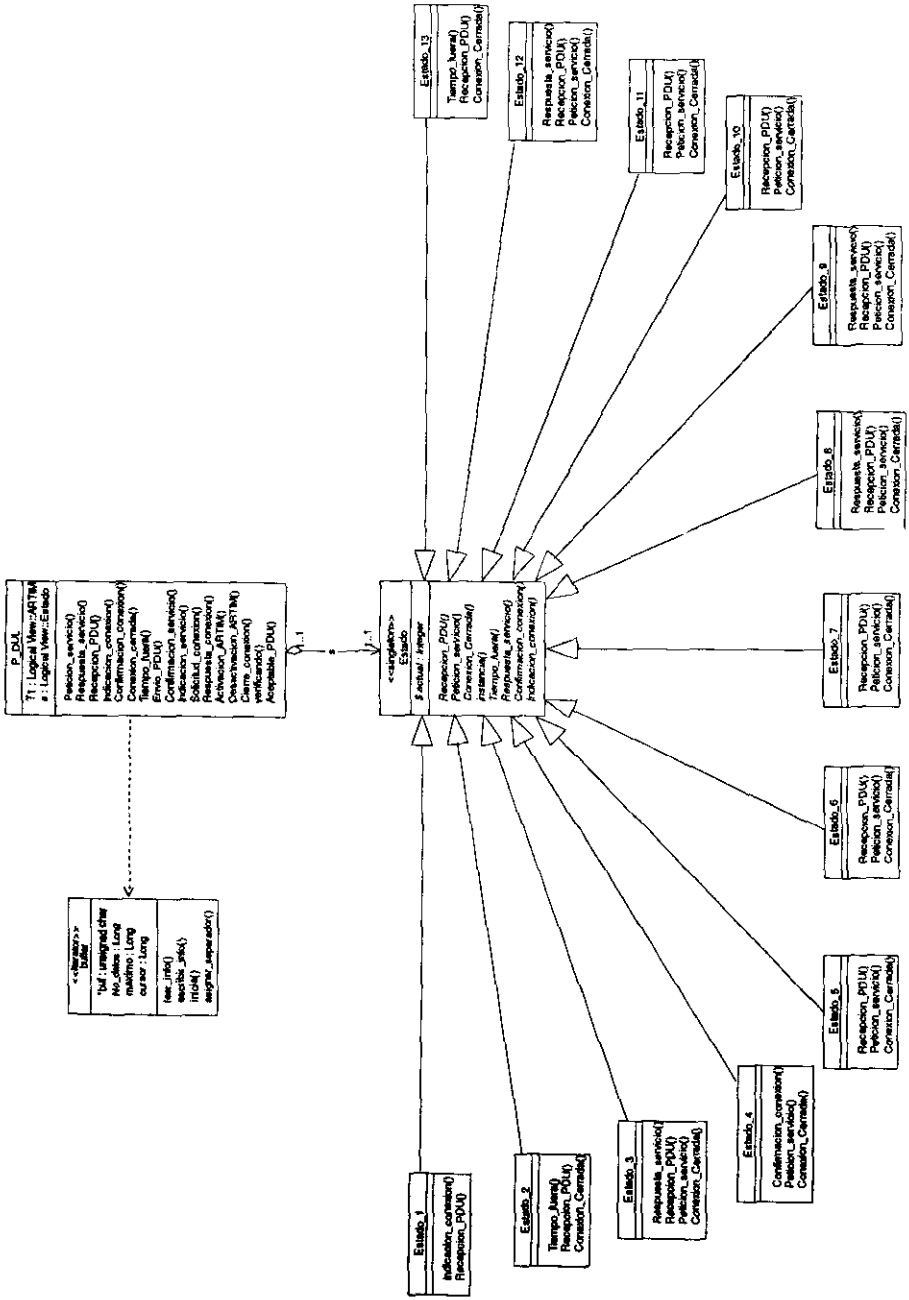


Figura 3.19: Estructura de clases para la máquina de estados finitos (MEF).

3.3.1.3. Patrón "Iterator"

El patrón "iterator" es definido [Gamma95] como un patrón que provee una forma de accesar secuencialmente los elementos de un objeto agregado (cola), sin exponer su representación básica y con la posibilidad de regresar siempre al primer elemento.

Los componentes compuestos tienen una cola básica en la que se guarda cualquier objeto heredado de la clase *Formato*. La operación para esa cola consiste en agregar elementos para después manipularlos. Para el caso de los PDU's compuestos, que pueden contener uno o varios elementos, simples o compuestos, se requiere recorrer en un orden específico la cola para que cada componente calcule su longitud después de haberse instanciado y, posteriormente, se vuelve a recorrer para serializar su información en un *buffer*, garantizando el orden de bytes apropiado. Para este requerimiento se utilizó el *TDA* cola al que se le introdujo el patrón "iterator" para manipularla en forma apropiada. Las colas utilizadas en este trabajo, se definieron a través de una clase *cola_iter* heredando del TDA *Cola*, ambas parametrizadas.

Por otro lado, este mismo patrón es utilizado en la estructura de una clase *buffer*. En este caso, se requiere leer y escribir información en forma secuencial. Sin embargo, cuando la información proviene de la interfase TCP es diferente que cuando proviene de una AE. Cuando la información llega de TCP, el PDU se identifica por el primer dato contenido en el buffer que representa su tipo y cuando llega de una AE, se utilizó por convención, un carácter separador para identificar a cada elemento almacenado en el buffer. De esta forma, el buffer debe ofrecer una funcionalidad apropiada como "iterator", en lectura y escritura, para manipular su información sin importar la forma en la que se encuentra almacenada.

3.3.2. Diseño de la máquina de estados finitos

Para el diseño de la máquina de estados finitos (MEF) se tomaron en cuenta algunas consideraciones relacionadas a la recepción de eventos, a la ejecución de acciones, al manejo del contador de tiempo fuera y a las diferentes fases de comunicación con TCP. La clase *P_DUL* debe encargarse de generar, validar y serializar los PDU's durante el proceso de recepción y envío. Además, al estar asociada con la *AE* y con la *Interfase_TCP*, la clase *P_DUL* se debe encargar de recibir los eventos y realizar las acciones, además de manipular el contador de tiempo fuera (*ARTIM*).

3.3.2.1. Manejo de eventos

Los eventos se procesan a través de las funciones de *P_DUL*, de acuerdo a las siguientes reglas:

- El protocolo ofrece cinco servicios a AE's a través de primitivas. Esto se traduce en cuatro eventos de *petición de servicios* y en tres eventos de *respuesta a servicios*, que se pueden agrupar en las funciones: *Petición_servicio* y *Respuesta_servicio.*
- El protocolo a través de su interfase con TCP puede recibir ocho diferentes PDU's (eventos), incluyendo PDU's desconocidos. La recepción de PDU's se realiza a través de una única función llamada *Recepción_PDU.*
- Tres eventos consideran la indicación de conexión, confirmación de conexión e indicación de cierre de conexión, a través de TCP, que se realizan con las funciones: *Indicación_conexión*, *Confirmación_conexión* y *Conexión_cerrada.*
- Cuando se excede el tiempo del controlador se produce un evento que invoca a la función *Tiempo_fuera.*

3.3.2.2. Manejo de acciones

Las acciones planteadas en la sección 3.1.2.2 se producen en respuesta a un evento, dependiendo del estado actual y se agrupan en las siguientes funciones de la clase P_DUL:

- *Envío_PDU*, para enviar los siete diferentes PDU's a través de TCP.
- *Confirmación_servicio*, que confirma los servicios A-ASSOCIATE y A-RELEASE a AE's.
- *Indicación_servicio*, para indicar los cinco servicios a AE's.
- *Solicitud_conexión*, para solicitar conexión a TCP.
- *Respuesta_conexión*, que responde a la conexión con TCP.
- *Activación_ARTIM* y *Desactivación_ARTIM*, para controlar el contador de tiempo ARTIM.
- *Cierre_conexión*, que cierra la conexión con TCP.

Acciones	Funciones de la clase P_DUL
AE-1	*Solicitud_conexion()*
AE-2	*Enviar_PDU(A_ASSOCIATE_RQ)*
AE-3	*Confirmacion_servicio(A_SSOCIATE)(aceptación)*
AE-4	*Confirmacion_servicio(A_SSOCIATE)(rechazo)* *Cierre_conexion()*
AE-5	*Respuesta_conexion()* *Activacion_ARTIM()*
AE-6	*Desactivacion_ARTIM()* *si Aceptable_PDU()* *Indicacion_servicio(A_ASSOCIATE)* *otro Envio_PDU(A_ASSOCIATE_RJ)* *Activacion_ARTIM()*
AE-7	*Envio_PDU(A_ASSOCIATE_AC)*
AE-8	*Envio_PDU(A_ASSOCIATE_RJ)* *Activacion_ARTIM()*

Tabla 3.1: Funciones que ejecutan las acciones relacionadas al establecimiento de asociación

Otra consideración en el diseño de la MEF es que sólo puede haber un estado activo a la vez.

Las funciones de la clase P_DUL que responden a las acciones definidas en la sección 3.1.2.2 del estándar se agrupan de acuerdo a las tablas 3.1, 3.2, 3.3 y 3.4.

Con base en la propuesta de [Sane95], [Ran93] y a los patrones documentados en [Gamma95], se plantea el uso de los siguientes patrones para la MEF.

3.3.2.3. Patrón "State"

A través de este patrón se definen objetos de la clase *Estado* que responden en forma distinta ante un mismo evento. La implementación de este patrón es similar a la propuesta de [Gamma95]. En este caso, se generan 13 subclases de la clase *Estado* y cada una de ellas define su respuesta de acuerdo al evento que recibe.

3.3.2.4. Patrón "Singleton"

Con el uso de este patrón, se asegura que una sola instancia de los posibles estados de la MEF está activa. Se define una función *Instancia*, perteneciente a la clase

Acciones	Funciones de la clase P_DUL
DT-1	*Envio_PDU(P_DATA_TF)*
DT-2	*Indicacion_servicio(P_DATA)*

Tabla 3.2: Funciones que ejecutan las acciones relacionadas a la transferencia de datos

Acciones	Funciones de la clase P_DUL
AR-1	*Enviar_PDU(A_RELEASE_RQ)*
AR-2	*Indicacion_servicio(A_RELEASE)*
AR-3	*Confirmacion_servicio(A_RELEASE)* *Cierre_conexion()*
AR-4	*Envio_PDU(A_RELEASE_RP)* *Activacion_ARTIM()*
AR-5	*Desactivacion_ARTIM()*
AR-6	*Indicacion_servicio(P_DATA)*
AR-7	*Envio_PDU(P_DATA_TF)*
AR-8	*Indicacion_servicio(A_RELEASE)(resuelve colision)*
AR-9	*Enviar_PDU(A_RELEASE_RP)*
AR-10	*Confirmacion_servicio(A_RELEASE)*

Tabla 3.3: Funciones que ejecutan las acciones relacionadas a la liberación de asociación

Acciones	Funciones de la clase P_DUL
AA-1	*Envio_PDU(A_ABORT)* *Activacion_ARTIM()*
AA-2	*Desactivacion_ARTIM()* *Cierre_conexion()*
AA-3	*si Aborto_iniciado()==usuario* *Indicacion_servicio(A_ABORT)* *Cierre_conexion()* *otro* *Indicacion_servicio(A_ABORT)*
AA-4	*Indicacion_servicio(A_ABORT)*
AA-5	*Desactivacion_ARTIM()*
AA-6	
AA-7	*Envio_PDU(A_ABORT)*
AA-8	*Envio_PDU(A_ABORT)* *Indicacion_servicio(A_P_ABORT)* *Activacion_ARTIM()*

Tabla 3.4: Funciones que ejecutan las acciones relacionadas al aborto de asociación

abstracta *Estado*, que garantiza la activación única de cada objeto generado de subclases.

Los patrones planteados para la MEF se expresan en las figuras 3.17 y 3.19. El mecanismo de la MEF involucra a dos objetos: uno de tipo *P_DUL* y otro de tipo *Estado*, activo en ese momento. Por ejemplo, cuando llega un PDU, este es recibido por el objeto de tipo *P_DUL* a través de del mensaje *Recepcion_PDU()*, quien se lo pasa al estado activo también mandando el mensaje *Recepcion_PDU()*. El estado activo ejecuta la acción y cambia de estado, informándole al objeto de tipo *P_DUL* el nuevo estado.

3.3.3. Interfases

En la mayor parte de nuestras representaciones a través de diagramas de clases y de secuencia, se ha considerado la existencia de dos interfases: para AE's y para TCP.

La interfase hacia TCP se ha considerado en forma explícita. De esta forma, tiene asignadas funciones para conexión, desconexión, escritura y lectura de datos y la revisión de estados de error. Estas funciones son consideradas para los casos en donde la conexión es activa, o pasiva. Con estas funciones representadas por la clase *Interfase_TCP*, el protocolo, representado por la clase *P_DUL*, puede lograr la comunicación vía TCP.

En el caso de las AE's, P_DUL ofrece esta interfase a través de las funciones *Peticion_servicio*, *Respuesta_servicio*, *Confirmacion_servicio* e *Indicacion_servicio*, que representan a las 4 primitivas necesarias para ofrecer culquiera de los 5 servicios. La información necesaria que fluye entre el protocolo y las AE's, se realiza a través de un objeto de la clase *buffer*, pero deben respetarse las normas de codificación ISO-8649, correspondientes a la definición de servicios ACSE.

3.3.4. Arquitectura propuesta

En el proceso de diseño se observaron dos estructuras independientes constituidas por los formatos de datos y la máquina de estados finitos. La arquitectura propuesta engloba ambos diseños y se presenta en las figuras 3.17, 3.18 3.19.

El modelo propuesto para el manejo de los formatos de datos (figura 3.18) fue implementado de la manera descrita en la sección 3.4. Para las pruebas de la máquina de estados finitos se construyeron los diagramas de secuencias para modelar el comportamiento del servicio A-ASSOCIATE (confirmado), tanto del lado del solicitante (figura 3.20) como del solicitado (figura 3.21), por considerarse como el más complejo y para realizar comparciones con las comprobaciones de la fase de análisis.

3.4. Evolución

A partir de las figuras 3.17 , 3.18 y 3.19, se puede observar en la arquitectura del protocolo DUL la clara separación que existe entre los formatos de datos y la MEF cuando únicamente existe una relación de uso entre ellas, teniendo el rol de cliente la MEF y de servidor los PDU's. De esta forma, se establece un bajo acoplamiento entre ambas partes y por lo tanto, se pueden hacer evolucionar por separado.

Para el caso de este trabajo, la parte correspondiente a los formatos de datos se hicieron evolucionar hasta una representación final, utilizando como herramienta el lenguaje de programación C++. Esto ha permitido la construcción de PDU's a partir de datos ficticios para la realización de pruebas de generación de PDU's, serialización, cálculo de longitudes y la representación "Big Endian" en orden de bytes para valores binarios.

La parte correspondiente a la MEF se encuentra en proceso de evolución. Sin embargo, en las pruebas planteadas a través de los diagramas de secuencia de las figuras 3.20 y 3.21 se ha observado un funcionamiento apropiado.

En la interfase hacia TCP se han planteado 2 opciones. La primera, corresponde al uso de las bibliotecas ACE (*Adaptive Communication Environment*) propuestas en [Schmidt93], [Schmidt95a], [Schmidt96]. Sin embargo, de acuerdo a las especificaciones en la sección 3.1.3, esta herramienta rebasa estos requerimientos. La ventaja que tendría su uso es que la implantación del protocolo se podría realizar en varias plataformas, debido a que las bibliotecas se adaptan a ellas al momento de compilarse.

La segunda opción, es utilizar la implementación propuesta para el patrón "state", planteado en [Gamma95], que es muy parecida a la propuesta para la MEF pero con menos estados.

3.5. Mantenimiento

El mantenimiento es considerado después que el producto se ha liberado. Sus actividades consisten en adaptar nuevos requerimientos a la aplicación o reparar algún problema surgido después de su liberación. En el caso de este trabajo, la implementación de la MEF se realizará en un futuro inmediato para liberar una primera versión. Por lo tanto, no se aplican estas actividades hasta este momento.

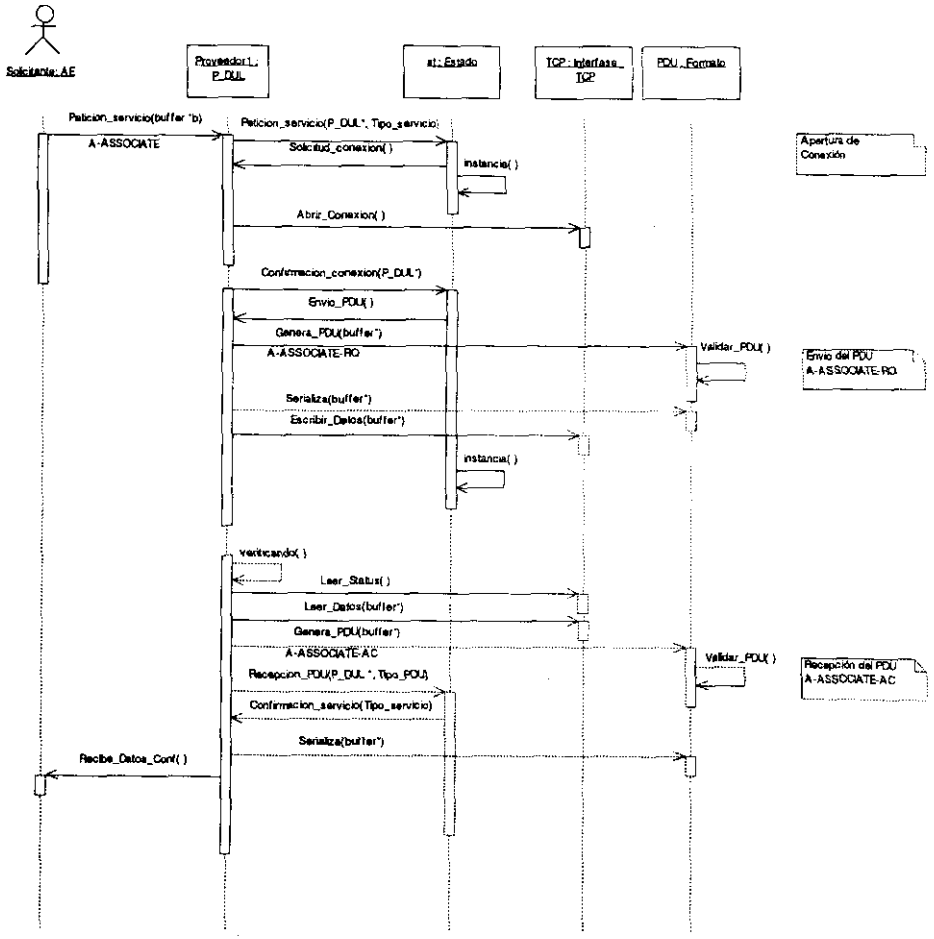


Figura 3.20: Prueba de una petición al servicio A-ASSOCIATE

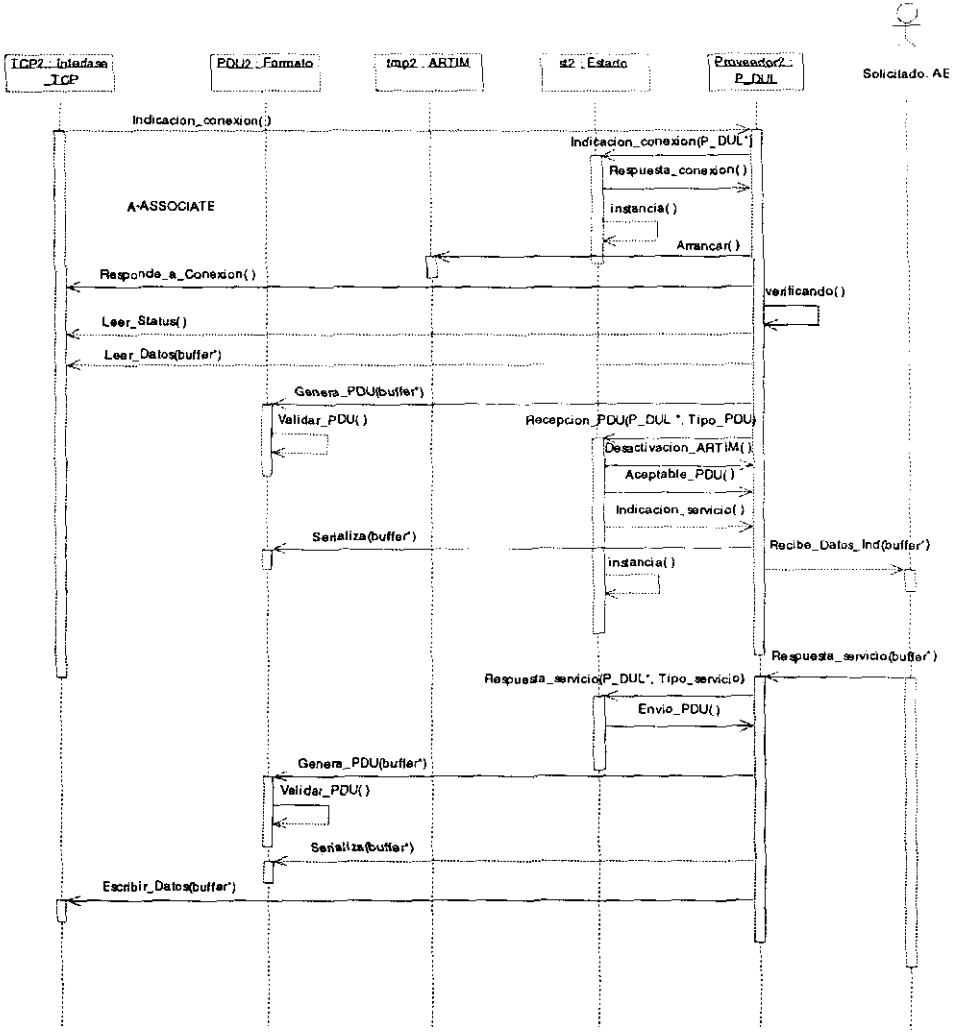


Figura 3.21: Prueba para una indicación al servicio A-ASSOCIATE

3.6. Conclusión y discusión

En este capítulo, al aplicar el proceso de desarrollo de software, descrito en [Booch96], se han presentado varios resultados correspondientes a sus diferentes fases. De esta forma, cada sección corresponde a una fase del macroproceso.

El proceso tiene como una de sus características importantes, que es iterativo e incremental. Sin embargo, como se mencionó al inicio, la descripción se hace en forma secuencial. Esto implica que en cada sección, se presentan los resultados obtenidos en la última iteración de cada fase del macroproceso.

También, se ha hecho énfasis en los productos que se obtienen, según lo establece la metodología. Véanse por ejemplo las secciones: 3.2.1, 3.2.2 y 3.2.3, para la fase de análisis, y la sección 3.3.4, para la fase de diseño. A su vez, la conclusión de cada fase se sustenta con las pruebas realizadas de acuerdo a las propuestas del mismo proceso.

En resumen, los resultados obtenidos en cada fase son:

- En la **fase de conceptualización**, la materia prima fueron las especificaciones del estándar, dando como resultado una visión global del protocolo.

- En la **fase de análisis**, el producto final es el modelo de dominio (figura 3.16) que se sustenta en las pruebas de comportamiento realizadas a través de los diagramas de secuencia mostrados en las figuras 3.14 y 3.15.

- En la **fase de diseño**, el producto final es la arquitectura propuesta para el protocolo que se sustenta en la eleción de patrones de diseño apropiados y se prueba con los diagramas de secuencia mostrados en las figuras 3.20 y 3.21, que hacen evidente la similitud en comportamiento planteado desde la fase de análisis (figuras 3.14 y 3.15).

- En la **fase de evolucíon**, el código propuesto generado a partir de la arquitectura, debe responder a los requerimiento planteados desde un principio. En el caso de este trabajo, se hicieron pruebas sólo al codigo generado para los formatos de datos.

4. Evaluación, resultados y discusión

Al revisar los productos de cada etapa del macroproceso, se presenta una evaluación de ellos tratando de describir en forma objetiva los beneficios obtenidos. Cabe mencionar que los productos a los que se llegó en cada etapa fueron la conclusión de varias iteraciones realizadas, aplicando el microproceso en algunas de ellas (análisis, diseño y evolución).

4.1. Conceptualización

La fase de conceptualización consistió básicamente en la revisión de las especificaciones del estándar DICOM, correspondientes al *soporte de comunicación en red para intercambio de mensajes* (parte 8 de [NEMA93]). Se consideró que las especificaciones definen los requerimientos para el protocolo. El producto obtenido consistió en entender las especificaciones para poder plantear los requerimientos. Sin embargo, en este ejercicio surgieron algunas dudas. En principio, el proceso establece que no necesariamente se debe entender el 100% de los requerimientos, aunque sí tenerlos presentes. Las dudas más importantes consistieron en:

- La correspondencia entre los parámetros recibidos por cada servicio y los datos involucrados en la creación de PDU's.
- La forma de instrumentar las primitivas asociadas con algún servicio.
- El comportamiento de los servicios más complejos (los confirmados).

4.2. Análisis

En la fase de análisis se obtuvo: un contexto del protocolo, un modelo de comportamiento y un modelo de dominio. Estos productos resuelven las dudas planteadas en la fase anterior, quedando claro todo el comportamiento relacionado al protocolo DUL. Los detalles para llegar a los resultados documentados en este trabajo son:

- Se logró agrupar las interacciones con el protocolo en seis casos de uso permitiendo identificar con claridad todas las posibles interacciones.

- Al modelar todas las interacciones con diagramas de secuencia (figuras 3.6 a la 3.13), incluyendo a los actores externos y el sistema (DUL), se establecieron las fronteras del protocolo con el exterior. De esta forma, quedó claramente establecido el conjunto de responsabilidades asignadas al protocolo que definen su contexto. Aclaramos que dichas responsabilidades se identifican en los diagramas con el texto que etiqueta a las líneas con flecha. La dirección de la flecha apunta al objeto al que se le asigna la responsabilidad.

- Ya teniendo definido el contexto del protocolo, se realizó un análisis para determinar cómo las responsabilidades asignadas a DUL se reparten hacia su interior. Para este propósito, se descubrieron las abstracciones que pudiesen participar en el comportamiento del protocolo, utilizando básicamente tarjetas CRC. Ya determinadas estas responsabilidades, se obtuvieron resultados de comportamiento modelando con diagramas de secuencia (figuras 3.14 y 3.15) dos de los escenarios más complicados, como son la petición de asociación del lado del solicitante y del solicitado, comprobando así su fucionamiento.

- Con el comportamiento ya modelado y comprendido, se planteó un modelo de dominio (figura 3.16), tomando como base las abstracciones definidas a través de tarjetas CRC y estableciendo relaciones entre ellas, a partir del modelo de comportamiento. Por ejemplo, se llegó a la conclusión que la relación entre la clase *MEF* y la clase *formato*, es una relación de dependencia (uso); la MEF va a construir y a utilizar un objeto de tipo formato, para destruirlo posteriormente cuando se requiera cambiar de formato por la demanda de un PDU diferente. De esta forma, se comprobó que el modelo de comportamiento responde a todas las demandas de operación hacia dentro del protocolo.

4.3. Diseño

En la fase de diseño, el producto principal es una arquitectura ejecutable, que pueda implementarse con las herramientas escogidas prácticamente en forma directa (análogo a la conversión de un algoritmo a código, después de haber sido refinado). Para lograr esta arquitectura, se utilizó la experiencia de terceros, plasmada en los patrones de diseño utilizados. Es importante señalar que el modelado

con diagramas de clases y diagramas de secuencia para esta fase, debe incluir el máximo detalle posible, expresando todos sus atributos y sus métodos (funciones miembro en C++), incluyendo los parámetros que deben manejar y el tipo de datos que deben regresar. La descripción de cómo se eligieron y se evaluaron los patrones de diseño, se plantea enseguida:

- Se partió del resultado de la fase anterior, teniendo ya una idea plena del comportamiento hacia el interior del protocolo y de su interacción con el exterior.

- Se retomaron estos comportamientos para adaptarlos de acuerdo a algún patrón de diseño. Por ejemplo, se sabía que en la MEF sólo podía estar activo un estado específico. Para poder realizar este comportamiento se utilizó el patrón "Singleton", cuya documentación así lo especifica y propone varias formas de implementarlo.

- Se pone mucha atención en estructuras y comportamiento que representen la suma de varios patrones de diseño. Por ejemplo, en el caso del comportamiento que deben expresar los formatos de datos, se plantea que un formato se puede construir en forma dinámica, sin saber específicamente el tipo, lo cual es resuelto por el patrón "Factory Method". Sin embargo, se plantea también una estructura de PDU's que puede ser simple o compuesta, que es formada por el patrón "Composite". En ambos casos, los patrones se expresan en el comportamiento y estructura de los PDU's.

- Teniendo ya una propuesta de arquitectura, como la que se muestra en las figuras 3.17, 3.18 y 3.19, se realizó una prueba (figuras 3.20 y 3.21), nuevamente para evaluar el comportamiento de los escenarios para la petición de asociación por parte del solicitante y del solicitado. Los primeros resultados de estas pruebas, sugirieron algunos cambios de refinamiento en la arquitectura. Por ejemplo, se llegó a la conclusión de que la información que recibe un objeto de *P_DUL* de las EA's o de la interfase con TCP, no debe pasarse al objeto actual de la clase *Estado*, teniendo como consecuencia, un cambio en la relación de dependencia de *P_DUL* hacia *Formato*, en lugar de *Estado* hacia *Formato*.

4.4. Evolución

- Para el caso de los PDU's, se realizó la implementación completa a partir de los patrones antes descritos y se probó la funcionalidad construyendo los PDU's A-ASSOCIATE-RQ y A-ASSOCIATE-AC por considerarse más

complejos. La prueba se dividió en cuatro etapas para simular el funcionamiento de la máquina de estados finitos de DUL.

1. Considerando la recepción de una primitiva de petición de asociación con datos ficticios, se generó un PDU A_ASSOCIATE_RQ a partir de ella. Una vez generado el PDU se desplegaron todos sus campos incluyendo las longitudes de cada componente, ya calculadas. Cabe señalar que varios de los campos especificados para cada PDU con sus componentes, son mandatorios y la información proveniente de la capa de aplicación que debe incluirse, es proporcionada por las primitivas asociadas al servicio.
2. Una vez generado el PDU, su información es serializada a través de un buffer para simular su paso a la capa de transporte, lo cual se realizó desplegando la información contenida en él, byte por byte y validando su contenido, especialmente la información binaria codificada en "Big Endian".
3. Para simular la recepción de información de la capa de transporte, se generó un PDU A-ASSOCIATE-AC a partir de un buffer, considerando que el buffer contiene la información recibida correspondiente al formato; después de crear el PDU, se desplegó su información para verificarla.
4. Para simular una respuesta a la capa de aplicación, se creó una primitiva ad-hoc en base al PDU A-ASSOCIATE-AC y se desplegó su contenido para validación.
5. La información manejada en la prueba incluyó varios elementos compuestos (Contextos de presentación), para complicar el manejo.

- La MEF no fue implementada; en un futuro inmediato se terminará y se probará en conjunto con los formatos de datos para tener concluido el protocolo DUL.

4.5. Discusión

Como puede inferirse de los párrafos anteriores, se obtienen varios beneficios al trabajar proyectos de software utilizando el proceso de desarrollo orientado a objetos, dentro de los que se encuentran:

- Se inicia con un conocimiento mínimo del sistema, que se va profundizando al aplicar las diferentes fases del macroproceso y microproceso, concluyendo

en una arquitectura refinada que se puede representar directamente con el lenguaje de programación elegido.

- Se puede tener la certeza de que la arquitectura obtenida es completa y flexible. Completa, porque las pruebas que se van aplicando en cada fase lo garantizan. Flexible, porque al suponer algún cambio en las especificaciones, en nuevas versiones del estándar, éstas se pueden modelar y adaptar con cierta facilidad a la arquitectura. Para el caso del protocolo, podría suponerse la inclusión de más formatos de datos o más estados a su MEF. Al modelar estos cambios se puede concluir que la flexibilidad de la arquitectura lo permite.

- Se aplica reuso a diferentes niveles. El reuso incluye el empleo de patrones de diseño, para reutilizar ideas de terceros, la aplicación del concepto de herencia en la definición de estructuras, al extender características de abstracciones y el reuso de código al utilizar bibliotecas ya desarrolladas.

Con respecto a la experiencia obtenida en la realización de este trabajo, se puede considerar:

- El manejo de diferentes niveles de abstracción, sin perderse en la cantidad de información manipulada (al menos al final del proyecto).

- La habilidad desarrollada para la selección de patrones de diseño apropiados y su adaptación específica.

- El entender con precisión el tipo de resultado que se debe obtener en cada fase de los subprocesos.

- La habilidad para encontrar la relación más apropiada entre abstracciones.

Reafirmando lo que se mencionó en la sección 1.5 correspondiente a los objetivos, no se ha encontrado una implementación de las especificaciones del estándar realizada con este tipo de metodología y, al parecer, tampoco se ha utilizado en algún producto comercial, por lo que fue uno de los motivos más fuertes para realizar el proyecto de esta forma.

5. Conclusiones y Perspectivas

De los resultados mencionados y discutidos en el capítulo anterior, podemos concluir que se han obtenido varios beneficios. Estos se traducen básicamente en: integrar al producto varias **características de calidad,** como consecuencia de desarrollar aplicaciones con metodología orientada a objetos, además de, **experiencia personal** en el desarrollo de este tipo de proyectos. Cabe mencionar que las características de calidad se hacen evidentes de los resultados obtenidos.

Como parte de las características de calidad identificadas, se pueden mencionar:

Confiabilidad y compatibilidad. Al ir estableciendo pruebas en cada fase, se puede asegurar un funcionamiento correcto y acorde con las especificaciones del protocolo. De esta forma, no habría duda en la comunicación a través de DUL con sistemas remotos.

Extensibilidad. Se logra al definir partes acopladas débilmente (MEF y PDU's) dentro del protocolo. Si llegaran a cambiar las especificaciones de DUL en otra versión de DICOM (por ejemplo, especificando más servicios para DUL), la MEF y los formatos pueden verse afectados al demandarse algunos cambios. Sin embargo, el acoplamiento débil entre ellos garantiza que los cambios en uno no afecten al otro en gran medida, lo cual facilita la extensión del protocolo.

Portabilidad. Al definir interfases completas y débilmante acopladas para la comunicación con TCP y entidades de aplicación DICOM. Las partes ya concluidas de DUL, se desarrollaron en plataforma UNIX utilizando como herramienta el lenguaje de programación C++. Si se requiere portar el protocolo a plataformas tipo PC con Windows como sistema operativo, las interfases para TCP siguen siendo las mismas y lo que cambia es la implementación de sus métodos (funciones miembro). Esto garantiza que las partes correspondientes al protocolo no sufran cambios y, como consecuencia, que exista una adaptación sencilla a la nueva plataforma.

Fácil de usar. Al definir una interfase simple con las entidades de aplicación para ofrecer sus 5 servicios a través de primitivas. Se definieron funciones específicas para solicitar, responder, indicar y confirmar servicios, en donde se considera que todos los parámetros son transferidos a través de un área de memoria o "buffer".

Dentro de la experiencia personal podemos considerar:

1. El aprender a manejar proyectos con la metodología orientada a objetos. En un principio, se presentó mucha incertidumbre al no entender varios aspectos del proyecto, debido a la cantidad de información involucrada en las especificaciones de DUL. Sin embargo, al ir afinando la aplicación de la metodología, se fue obteniendo gradualmente un mayor control.

2. El aprender a establecer fronteras conceptuales a nivel de objetos, módulos y de la aplicación misma, ofreció un gran avance en las actividades de abstracción.

3. El sustentar desde diferentes perspectivas las relaciones entre abstracciones, ofreció un avance en el desarrollo de esta actividad.

4. El aprender a identificar patrones de acuerdo al comportamiento en diferentes partes del protocolo y a las estructuras mostradas, ofreció la posibilidad de agilizar la fase de diseño, al activar mecanismos de reuso con patrones de diseño, ofreciendo soluciones "elegantes" y confiables.

5. El aprender a utilizar herramientas CASE como "*Rational Rose 98, de Rational Software Corporation*" para modelar con UML y "*CRC Tool, de ixtlan software*" para tarjetas CRC, ofreció la posibilidad de agilizar el proceso y documentarlo en forma automatizada.

6. El aprender a plantear pruebas para los productos obtenidos en las diferentes fases del proceso, ayudó a establecer seguridad en las actividades realizadas.

En un futuro inmediato, se concluirá la implementación del protocolo DUL para ser acoplado con las otras dos partes de las especificaciones DICOM: los protocolos *DIMSE para intercambio de mensajes* y las *entidades de aplicación DICOM*, que se están trabajando en forma paralela utilizando la misma metodología.

El objetivo final es un proyecto más ambicioso que pretende la creación de un conjunto de bibliotecas para generar aplicaciones relacionadas a la visualización,

almacenamiento, procesamiento y compresión de imágenes médicas digitales. Algunas de éstas se están trabajando también en forma paralela y se planea a mediano plazo poder ofrecerlas a alguna institución del sector salud.

A. Abreviaturas

AAPM: American Association of Physicists in Medicine

ACE: Adaptive Communication Environment

ACR: American College of Radiology

ACSE: Association Control Service Element

AE: Aplication Entity

ANSI: American National Standards Institute

ARTIM: Association Request/Reject/Release Timer

ASE: Aplication Service Element

CASE: Computer-Aided Software Engineering

CEN: Comité Européen de Normalisation

CMM: Capability Maturity Model

CR: Computed Radiography

CRC: Class Responsability Colaborators

CT: Computed Tomography

DICOM: Digital Imaging and Communications in Medicine

DIMSE: DICOM Message Service Element

DPCM: Differential Pulse Code Modulation

DSA: Digital Subtraction Angiography

DUL: DICOM Upper Layer

GIF: Graphics Interchange Format/CompuServe

HIS: Hospital Information System

IE: Information Entity

IOD: Information Object Definition

ISO: International Organization for Standarization

ISO/OSI: International Organization for Standardization/Open Systems Interconection

JIRA: Japanese Industry Radiology Apparatus

JPEG: Joint Photographics Expert Group

MEF: Máquina de Estados Finitos

MPEG: Motion Pictures Expert Group

MR: Magnetic Resonance

NEMA: National Electrical Manufacturers Association

NM: Nuclear Medicine

OMG: Object Management Group

OMT Object Modeling Technique

OSI: Open Systems Interconnection

PACS: Picture Archiving and Communication Systems

PDU: Protocol Data Unit

PDV: Presentation Data Value

PSDUL: Proveedor de Servicio DUL

RIS: Radiology Information System

TCP/IP: Transmission Control Protocol/Internet Protocol

TDA: Tipo de Dato Abstracto

UDP: User Datagram Protocol

UML: Unified Modeling Language

XR: X Ray

BIBLIOGRAFÍA

[Allen92] Allen L. & Frieder O.; "*Exploiting Database Technology in the Medical Arena*"; IEEE Engineering in Medicine and Biology, Marzo 1992, pg. 42-49.

[Ambler97] Ambler S. W.; "*The Unified Modeling Language v1.1 and Beyond: The Techniques of Object-Oriented Modeling, A White Paper*"; (*http://www.ambysoft.com/umlAndBejond.pdf*) Septiembre 1997.

[Ambler97a] Ambler S. W.; "*CRC Modeling: Bridging the Communication Gap Between Developer and Users, A white paper*"; (*http://www.ambysoft.com*), Julio 1997.

[Ambler97b] Ambler S. W.; "*The Object-Oriented Modeling Process: An Architecture-Driven Approach, A white paper*"; (http://www.ambysoft.com/ooModelingProcess.pdf) Agosto 1997.

[Bergner97] Bergner K., Andreas R. & Sihling M.; "*Using UML for Modeling a Distributed Java Application*"; Mathematisches Institut und Institut Für Informatik Technische Universität München, TUM-I9735, (*http://www4.informatik.tu-muenchen.de*) Julio 1997.

[Booch94] Booch G.; *Object Oriented Design with aplications*; Second Edition, The Benjamin Cummins Publishing Company, 1994.

[Booch96] Booch G.;*Object Solutions, Managing the Object-Oriented Project*; Addison Wesley, 1996.

[Booch97] Booch G., Rumbaugh J. & Jacobson I.; *Unified Modeling Language, version 1.0*; Rational Software Corporation, Sta. Clara California (USA), (*http://www.rational.com*) Enero 1997.

[Booch99] Booch G., Rumbaugh J. & Jacobson I.; *The Unified Modeling Language User Guide*; Addison Wesley, 1999.

[Clunie95] Clunie D.; *Medical Image Format FAQ*; (*http://www.rahul.net/pub/dclunie/medical-image-faq/html/*).

[Coad92] Coad P. & Yourdon E.; *Object Oriented Design*; Prentice Hall, 1992.

[Comer95] Comer E.; *Internetworking with TCP/IP, volume I: Principles, Protocols and Architecture*; Prentice Hall, 1995.

[Comer96] Comer D. E. & Stevens D. L.; *Internetworking with TCP/IP, volume III: Client-Server programming and aplications*; Prentice Hall, 1996.

[Coplien94] Coplien J. O; "*Software Design Paterns: Common Question and Answers*"; in Proceedings of Object Oriented Expo New York, pg. 39-42, Junio 1994, New York SIGS Publications.

[Coplien97] Coplien J. O; "*Idioms and Patterns as Architectural Literature*"; IEEE Software Special Issue on Objects, Patterns, and Architectures, Enero 1997.

[Dale88] D'Alessandro M; "*Computers in Radiology: The totally Digital Radiology Department of the Future*"; SIGBIO Newsletter, Vol. 10, No. 4, 1988 pg. 2-6.

[Gamma93] Gamma E., Helm R., Johnson R., Vlissides J.; "*Design Patterns: Abstraction and Reuse of Object Oriented Design*"; Proceedings of ECOOP'93, Kaiserslautern Alemania 1993.

[Gamma95] Gamma E., Helm R., Johnson R., Vlissides J.; *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison Wesley, 1995.

[Hintel94] Hintel R.; *Implementation of the DICOM 3.0 Standard, A Pragmatic Handbook*; RSNA, 1994.

[Holzmann91] Holzmann G. J; *Design and Validation of Computer Protocols*; Prentice Hall, 1991.

[Horiil95] Horiil S., Prior W., Bidgood W., Parisot C., Claeys G.; "*DICOM: An Introduction to the Standard*"; (*http://xray.hmc.psu.edu*).

[Humphrey89] Humphrey W. S; *Managing the software process*; Addison Wesley 1989.

[Jacobson99] Jacobson I., Booch G. & Rumbaugh J. ; *The Unified Software Development Process*; Addison Wesley, 1999.

[Jensch98] Jensch P, Eichelberg M., Riesmeier J., Gehlen S., Hewett A. & Barth A.; *DICOM OFFIS Software*; (*http://offis.offis.uni-oldenburg.de/projekte/dicom/dicom_software_e.html*).

[Khare95] Khare R; "*On the diffusion of Christopher Alexander's A Pattern Language into Software Architecture*"; (*http://xent.w3.org/~khare/Alexander.html*) Abril 1995.

[Larmouth94] Larmouth J; *Understanding OSI*; University of Salford, 1994.

[Lea94] Lea D.; "*Christopher Alexander: An Introduction for Object-Oriented Designers*"; ACM SIGSOFT, Software Engineering Notes, Vol 19, No. 1, p 39, Enero 1994.

[Leotta93] Leotta D F & Kim Y; "*Requirements for Picture Archiving And Communications*"; IEEE Engineering in Medicine and Biology, Marzo 1993, pg. 62-69.

[Monarchi92] Monarchi D. E. & Puhr G. I.; "*A Research Typology for Object-Oriented Analysis and Design*"; Communications of the ACM, Vol. 35, No. 9, Septiembre 1992, pg. 35-47.

[Mun93] Mun S. K, Freedman M, Kapur R; "*Image Management And Communications For Radiology*"; IEEE Engineering in Medicine and Biology, Marzo 1993, pg. 70-80.

[NEMA93] NEMA; "*Digital Imaging and Comunications in Medicine (DICOM)*"; NEMA Standards Publications PS 3.1 to PS 3.9, 1993.

[Orozco92] Orozco-Barbosa L., Karmouch A., Georganas N. D. and Goldberg M.; "*A Multimedia Interhospital Communications System for Medical Consultations*"; IEEE Journal on Selected Areas in Communications, Vol 10, No. 7, Septiembre 1992, pgs. 1145-1157.

[rfc791] Postel J. (Editor); "*Internet Protocol: DARPA Internet Program Protocol Specification*", *RFC 791*; USC/Information Sciences Institute, Septiembre 1981.

[rfc793] Postel J. (Editor); "*Transmission Control Protocol: DARPA Internet Program Protocol Specification*", *RFC 793*; USC/Information Sciences Institute, Septiembre 1981.

[Rago93] Rago S. A.; *Unix System V Network Programming*; Adison Wesley, 1993.

[Ramam96] Ramamoorthy C. V. & Tsai W.; "*Advances in Software Engineering*"; IEEE-Computer, Vol. 29, No. 10, octubre, 1996, pgs. 47-58.

[Ran93] Ran S. A.; "*Pattern of Events*" *en Pattern Languages of Program Design*; Addison Wesley, 1995.

[Rimmer90] Rimmer S.; *Bit-Mapped Graphics*; Windcrest Books, 1990.

[Rumbaugh99] Rumbaugh J., Jacobson I. & Booch G.; *The Unified Modeling Language Reference Manual*; Addison Wesley, 1999.

[RSNA93] *Documentation of the Radiology Society of North America (RSNA)*, 1993 DICOM Demostration.

[Sane95] Sane A. & Campbell R.; "*Object-Oriented State Machines: Subclassing, Composition, Delegation, and Genericity*"; OOPSLA'95.

[Schmidt92] Schmidt D. C.; "*System Programming with C++ Wrappers: Encapsulating Interprocess Communication Mechanisms with Object-Oriented Interfaces*"; C++ Report Magazine, Septiembre/Octubre 1992.

[Schmidt92a] Schmidt D. C.; "*IPC SAP: A Family of Object-Oriented Interfaces for Local and Remote Interprocess Communication*"; C++ Report Magazine, Noviembre/Diciembre 1992.

[Schmidt93] Schmidt D. C.; "*The Adaptive Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Sotware*"; $11^{th}$and 12^{th} Sun User Group Conferences in San Jose, California Diciembre 7-9, 1993 y San Francisco, California, Junio 14-17, 1993.

[Schmidt95] Schmidt D. C.; "*Reactor: An object behavioral pattern for current event demultiplexing and event handler dispatching*", en *Pattern Languages of Program Design*; Addison Wesley, 1995.

[Schmidt95a] Schmidt D. C.; "*Acceptor: A Design Pattern for Passively Initializing Network Services*"; C++ Report Magazine, Noviembre/Diciembre 1995.

[Schmidt96] Schmidt D. C.; "*Connector: A Design Pattern for Actively Initializing Network Services*"; C++ Report Magazine, Enero 1996.

[Smith94] Smith P. & BSG; *Client-Server Programming and Aplications*; Volumen III, Prentice Hall, 1994.

[Vanzyl95] Vanzyl A.; *Increasing Web Bandwidth through Image Compression: An Overview of GIF, JPEG and Fractal Compression Techniques*; Monash University, (*http://www.monash.edu.au/informatics/Default.html*).